Supercomputing Frontiers and Innovations

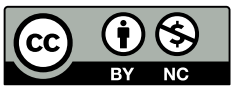
2016, Vol. 3, No. 2

Scope

- Enabling technologies for high performance computing
- Future generation supercomputer architectures
- Extreme-scale concepts beyond conventional practices including exascale
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Distributed operating systems, kernels, supervisors, and virtualization for highly scalable computing
- Scalable runtime systems software
- Methods and means of supercomputer system management, administration, and monitoring
- Mass storage systems, protocols, and allocation
- Energy and power minimization for very large deployed computers
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Parallel performance and correctness debugging
- Scientific visualization for massive data and computing both external and in situ
- Education in high performance computing and computational science

Editorial Board

Contents



Foreword to the Special Issue of International Journal of Supercomputing Frontiers and Innovations

This special issue of International Journal of Supercomputing Frontiers and Innovations contains selected papers from the second conference Supercomputing Frontiers 2016 which took place in Singapore on March 1518, 2016 (http://supercomputingfrontiers.com/2016/).

This year we continued with the goal set for the first conference in 2015: *"to bring some of the most highly recognized Supercomputing authorities to our community of users to expose our community to the past achievements and to the visionary ideas of those exceptional trendsetters"*. But gradually we also wish to give back to the supercomputing community with quality contributions from the local researchers based in Singapore. An example of this local content are the two papers in the current issue: the one by K. Ban et. al. on InfiniCloud 2.0, and the second by John Gustafson entitled: *"A Radical Approach to Computation with Real Numbers"*. There was also a special session on the global InfiniCortex project which was initiated and is being run from A*CRC in Singapore. The remaining papers in this issue were contributed by authors from the USA, France and Russia.

The conference program featured an exciting line-up of speakersand included sessions on:

1. Supercomputing applications in domains of critical impact and especially those requiring computer resources approaching Exascale;
2. Efforts to build exascale supercomputers;
3. Exascale storage and file systems;
4. New, non-standard processor architectures including neuromorphic processors and automata processors;
5. Convolution of supercomputing, artificial intelligence and the biological brain;
6. Languages for exascale and for human-computer interactivity;
7. InfiniCortex session; and
8. The country reports from India, Japan, China and Singapore.

The keynote speakers at this years Supercomputing Frontiers 2016 were: Srinivas Aluru (Georgia Tech, USA), Baroness Susan Greenfield (Oxford Univeristy, UK), Horst Simon (Lawrence Berkeley National Laboratory , USA) and Bronis de Supinski (Lawrence Livermore National Laboratory, USA). The meeting brought together over 330 participants, featured 17 invited speakers and 27 contributed talks. There were also 8 workshops attended by over 200 participants. I encourage the reader to consult the 2016 conference program for the full line up of the speakers and topics of all talks, since only a small selection is presented here.

Finally, let me rephrase what I said in my foreword to Vol.2 No.3 special issue of IJSF&I: Supercomputing Frontiers 2017 will be held on March 1317, 2017, in Singapore for its third edition. We promise a truly outstanding program with excellent scientific content. *We welcome your participation!*

Marek Michalewicz
Chairman of the Scientific Programme Committee
and Chairman of the Organising Committee

DOI: 10.14529/jsfi160201

A Dynamic Congestion Management System for InfiniBand Networks

Fabrice Mizero[1]***, Malathi Veeraraghavan***[1]***, Qian Liu***[2]***, Robert D. Russell***[2]***, John M. Dennis***[3]

While the InfiniBand link-by-link flow control helps avoid packet loss, it unfortunately causes the effects of congestion to spread through a network. Even flows that do not pass through congested ports can suffer from reduced throughput. We propose a Dynamic Congestion Management System (DCMS) to address this problem. Without per-flow information, the DCMS leverages performance counters of switch ports to detect onset of congestion and determines whether-or-not victim flows are present. The DCMS then takes actions to cause an aggressive reduction in the sending rates of congestion-causing (contributor) flows, if victim flows are present. On the other hand, if there are no victim flows, the DCMS allows the contributor to maintain high sending rates and finish as quickly as possible. The value of dynamic management of a switch congestion-control parameter called `Marking_Rate`, which is responsible for how quickly contributor flows can be throttled, is evaluated in an experimental testbed. Our results show that dynamic congestion management can enable a network to serve both contributor flows and victim flows effectively. The DCMS solution operates within the constraints of the InfiniBand Standard.

Keywords: InfiniBand, Congestion control, Link-by-link flow control, Cascading rate reductions, Dynamic parameter setting.

Introduction

InfiniBand (IB) is widely used in high performance computing (HPC) systems. Among other factors, InfiniBand owes its growing adaptation to its high link rates, low latency and low packet loss. The InfiniBand protocol specification supports a credit-based link-by-link flow control to avoid packet loss and a congestion control system based on explicit congestion notifications.

As noted in prior work [1], the presence of link-by-link flow control causes the effects of congestion to spread backwards in the network. When an output port P1 of a switch becomes congested, the input-side buffer of another port P2 on the same switch, through which a congestion-causing (contributor) flow enters the switch, will fill up causing a reduction in the rate at which flow-control credits are granted by port P2 to port P3 of an upstream switch. This causes a reduction in the effective rate of port P3. Such a rate reduction could cascade backwards and reduce the effective rates of many ports. Bulk-data flows passing through victim ports (ports with reduced effective rates) become victim flows (suffer reduced throughput), even though their own paths do not traverse the congested port. The problem statement of this work is to address the spreading effects of congestion.

We propose a dynamic Congestion Management System (DCMS) that (a) monitors switch-port counters to determine if victim flows have been created by a congestion event, and (b) if there are victim flows, the DCMS dynamically modifies a switch congestion-control parameter to dissipate the congestion event rapidly. A key consideration is the tradeoff between the creation of victim flows vs. a reduction in the throughput of contributor flows.

Experiments were conducted on a two-switch, multi-host InfiniBand testbed. First, the impact of switch congestion-control parameters were studied to determine the default settings

[1]University of Virginia; {fm9ab, mv5g}@virginia.edu
[2]University of New Hampshire; {qga2, rdr}@unh.edu
[3]National Center for Atmospheric Research; {dennis}@ucar.edu

that would allow contributor flows to enjoy high throughput as long as no victim flows are created. Next, our DCMS proof-of-concept prototype was executed and a switch congestion-control parameter was modified dynamically in a manner that caused senders of contributor flows to reduce their packet injection rates if the DCMS detected the presence of victim flows. When victim flows and/or contributor flows that created victim ports end, or a duration threshold is crossed, the DCMS resets the switch congestion-control parameter back to its default setting.

The novelty of this work lies in our proposal of a dynamic congestion management system (DCMS). The solution is InfiniBand compliant in that the DCMS works in conjunction with off-the-shelf switches requiring no modifications. The importance of dynamic parameter control to enable the network to serve both contributor and victim flows is demonstrated through experiments.

Section 1 offers the reader background on InfiniBand flow control and congestion control. Section 2 describes the spreading effects of a congestion event using a new approach based on a concept of cascading rate reductions. Section 3 describes the DCMS algorithm. Section 4 describes our experiments with a DCMS prototype. Related work is reviewed in Section 5, and Section 5 presents our conclusions.

1. Background

The InfiniBand protocols [2] include link-layer flow control and transport-layer congestion control. Link-layer flow control is used to ensure 0 packet loss due to buffer overflows. A transmitter is permitted to send packets onto a link only when it has received sufficient credits to do so from the receiving end of the link. A Flow Control Packet (FCP) is sent from the receiving side to the transmitting side of a link to explicitly provide information on the amount of space left in the receive buffer.

The transport-layer congestion control is based on Explicit Congestion Notification (ECN), wherein once congestion is detected at a switch, the contributing sources are notified to reduce their packet injection rates. Coordinated actions are required at the (i) switch that detects congestion on one of its ports, (ii) destination Host Channel Adapters (HCAs) of flows that traverse the congested port, and (iii) source HCAs of those flows.

Specific details of how a switch decides that one of its ports is congested are left to vendor implementation. In one approach described by Gran and Reinemo [3], when the fill-ratios of input-port buffers holding packets destined to a particular output port exceed a set threshold, the switch will consider the output port to be congested. A parameter called `Threshold` controls how quickly a switch reacts to congestion, with a value 15 indicating the fastest reaction to congestion onset, and a value 0 for disabled congestion control. The switch then sets a bit called Forward ECN (FECN) in the transport-layer header to 1 for a fraction of the packets transmitted onto the output port. The value of the fraction is determined by a configurable switch parameter called `Marking_Rate`. The `Marking_Rate` is the mean number of unmarked packets sent between consecutive marked packets, where "marking" refers to the setting of the FECN bit. Therefore, the higher the `Marking_Rate`, the lower the rate of generation of FECNs.

When a destination HCA receives a marked packet, the HCA sets the Backward ECN bit (BECN) in an Acknowledgment (ACK) or a data-carrying packet sent from the destination to the source, or the destination HCA generates an explicit Congestion Notification Packet (CNP) to the source of the flow.

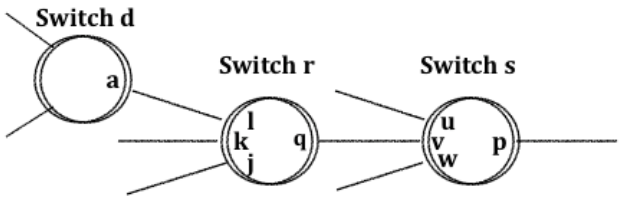


Figure 1. Illustrative InfiniBand network

When a source HCA receives a BECN-marked or CNP for a flow, the source HCA reduces the packet injection rate for that particular flow in the following manner. One Congestion Control Table (CCT) is maintained per port, and one CCT Index (CCTI) can be maintained per flow (queue-pair) or Service Level (SL). A queue-pair is comparable to TCP source and destination port numbers, while SL is a parameter that allows applications to indicate the desired service type. Each entry in the CCT specifies a packet injection delay number, which means that the injection delay between packets for a flow is determined by the CCTI associated with the flow. The HCA maintains two configurable parameters, `CCTI_Increase` and `CCTI_Timer`, per port per SL. When a source HCA receives a BECN-marked packet or CNP from the destination for a particular flow, it increases the CCTI of the flow by the `CCTI_Increase` value for the corresponding port/SL. On every reset of the `CCTI_Timer`, the CCTI of all flows associated with that port/SL are decreased by 1.

In summary, multiple configurable parameters in switches and in HCAs jointly determine how quickly congestion is detected, and how quickly the sending rates of contributor flows are throttled or restored. Throttling contributor flows could have a positive impact on victim flows. On the other hand, throttling contributor flows could have the negative impact of lowering the throughput of these flows.

The purpose of this work is to develop and evaluate schemes that can manage the above-described tradeoff through dynamic modifications to the Congestion-Control (CC) parameters at switches. The only HCA CC parameter that is set on a per-flow basis is CCTI, and therefore it could potentially be modified dynamically. However, it is complex to deploy an external management server that can determine per-flow characteristics and then take actions to dynamically modify the CCTI of a flow at its source. In IP networks, NetFlow and other similar mechanisms are built into routers to reconstruct flow characteristics from sampled or unsampled packets. To our knowledge, there is no comparable feature in InfiniBand switches, which makes it more complex to develop external solutions for flow reassembly.

2. Causes, modes, and effects of congestion

This section describes the causes of congestion, two different modes of congestion, and the effects of congestion.

Congestion occurs at a port p of a switch s when the aggregate arrival rate of packets destined to port p exceeds its capacity. This is the main *cause* of congestion, and port p of switch s is referred to as the *root port* of a congestion event. For example, consider the network shown in Fig. 1. If the aggregate rate of packets arriving at ports u, v, and w, and destined to port p of switch s, exceeds the capacity of port p, then port p will enter a state of congestion.

Formally, we say that port p of switch s is congested at time t if

$$\sum_{v \in \mathbf{P}_s} f_{svp}(t) > C_{sp}, \tag{1}$$

where $f_{svp}(t)$ is the rate of arrival of packets at port v of switch s that are destined to port p of switch s, $\mathbf{P}_s$ is the set of all ports of switch s, and C_{sp} is the link capacity of port p of switch s. When congestion occurs at port p of switch s, input-side buffers of ports with incoming packets that are destined to port p will start filling up. If the configured thresholds are crossed the switch will detect the congestion event and take the actions described in Section 1.

There are two *modes* of congestion as illustrated by an example in the network of Fig. 1. Assume that the port p of switch s enters the state of congestion. If there is a flow that traverses ports j and q of switch r and ports v and p of switch s, then the buffer on the incoming side of port v of switch s will start to fill up, if the instantaneous arrival rate of packets on this flow exceeds the share of the congested-port p bandwidth available to this flow. When this buffer fills up, the rate at which FCPs are generated by port v of switch s to port q of switch r to offer the latter credits for packet transmission will decrease. Effectively, the rate of port q of switch r becomes reduced.

Formally, if port p of switch s enters congestion at time t, then at time $t + \varepsilon$, where ε is a small interval (on the order of nanoseconds in high-rate InfiniBand links),

$$R_{rq}(t + \varepsilon) < C_{rq}, \tag{2}$$

where $R_{rq}(t + \varepsilon)$ is the rate at which data is sent by the transmitter at port r of switch q at time $(t + \varepsilon)$, and port q of switch r is connected to some port v of switch s. Thus, in our usage, "rate" of a port is time-variant, while "capacity" of a port is time-invariant.

A reduction in the sending rate of a switch port can have cascading effects at neighboring switches. In the above example, the presence of a flow traversing port q of switch r and the congested port p of switch s is the cause of rate reduction of port q of switch r. Further, if a flow enters switch r at port l and is destined for port q, then packets from this flow will be served at a lowered rate by port q. Consequently, the input buffer at port l could fill up, causing FCPs to grant credits at a lowered rate to port a of switch d. This causes a rate reduction of port a of switch d. Events that cause these types of "Cascading Rate Reductions" are referred to as *CRR congestion events*. Ironically, this kind of cascading rate reductions occurs because of IB's zero-packet loss policy, which is enforced by the link-by-link flow control. In Ethernet networks, as there is no link-by-link flow control, a transmitter can simply send packets. If a switch buffer is full, packets will simply be dropped. Therefore, a congestion event is handled locally, and its effects do not spread to other switches, as with cascading rate reductions in InfiniBand networks.

On the other hand, if there was no flow passing through neighboring switch ports that also pass through the congested port, then the congestion event could be localized in that all rate reductions are limited to HCA ports. If switch ports are not affected, CRRs will not occur. For example, assume that three hosts are connected to ports u, w, and p of switch s. If the hosts connected to u and w initiate flows destined to the host connected to port p, then port p could get congested. This may in turn cause FCPs to limit credits to the HCAs at hosts connected to ports u and w. However, the rate reductions of these HCA ports cannot propagate to other ports, and therefore, we refer to this mode of congestion as *localized*.

The main *effect* of congestion is the creation of victim flows. A "victim flow" is a flow whose path does not traverse a congested port, and yet (i) does not enjoy the full spare capacity on its path, or (ii) its packets are subject to additional delays. Flows that share one or more links with contributor flows can become victim flows. Consider a flow V in our above example that traverses ports k and q of switch r and ports v and w of switch s. Assuming all links in our example network are Single Data Rate (SDR) links (i.e., 10 Gbps), but the inter-switch link is a Quad Data Rate (QDR) link (i.e., 40 Gbps), the flow V should enjoy SDR rate, if there are no other flows sharing its links. However, because the rate of the QDR inter-switch link will be determined by the rate at which FCPs grant credit to port q of switch r to send packets, the rate of flow V will be limited to this FCP-dictated rate rather than the spare capacity on its path. Hence flow V could become a victim flow.

In summary, congestion is *caused* by a packet arrival rate that exceeds link capacity. There are two *modes* of congestion: CRR and localized congestion. Both modes of congestion could have the *effect* of creating victim flows, which could be delay-sensitive flows or bulk-data flows.

3. Dynamic Congestion Management Solution (DCMS)

Table 1. Notation

Symbol	Meaning
s	switch index
p	port index
$\mathbf{S}$ and $\mathbf{P}_s$	Set of all switches, and all ports on switch $s \in \mathbf{S}$, respectively
$\mathbf{H}$ and $\mathbf{P}_h$	Set of all hosts, and all HCA ports on host $h \in \mathbf{H}$, respectively
$\mathbf{I}$ and $\mathbf{J}$	Set of all switch ports $\mathbf{I} = \bigcup_{s \in \mathbf{S}} \mathbf{P}_s$ and set of all host ports $\mathbf{J} = \bigcup_{h \in \mathbf{H}} \mathbf{P}_h$
$N : (\mathbf{S} \times \mathbf{I}) \rightarrow \{(\mathbf{S} \times \mathbf{I}) \cup (\mathbf{H} \times \mathbf{J})\} \cup \emptyset$	Mapping function that shows the neighbor switch/host and port to which each switch's port is connected, if present; $N(s,p) =$ null if $p \notin \mathbf{P}_s$
$W(s,p,t)$	PortXmitWait counter value of port $p \in \mathbf{P}_s$ at time t
$D(s,p,t)$	PortXmitData counter value of port $p \in \mathbf{P}_s$ at time t
$C(s,p,t)$	PortXmitCongTime counter value of port $p \in \mathbf{P}_s$ at time t
$\Delta X(s,p,t)$	$X(s,p,t) - X(s,p,(t-\tau))$, where $X \in \{W,D,C\}$, where τ is the inter-sweep interval
$\mathbf{V}(s,p)$	Set of victim ports (r,q) for which $\{(s,p),(r,q)\} \in \mathbf{O}_1$
$\mathbb{M}(s,p)$	`Marking_Rate` of port $p \in \mathbf{P}_s$ of switch s; {`Low`, `Default`, `High`}
$\mathbb{S}(s,p)$	State of $p \in \mathbf{P}_s$ of switch $s \in \mathbf{S}$; {`Low-MR`, `Default-MR`}
$\mathbb{I}(s,p)$	Interval count for low value of $\mathbb{M}(s,p)$
T_C	Congestion threshold
T_W	Rate-reduction threshold
T_D	Utilization-change threshold
T_I	Low-MR (`Marking_Rate`) duration threshold

We propose a Dynamic Congestion Management System (DCMS) to modify the `Marking_Rate` parameter in switches to cause reductions in the sending rates of contributor flows if there are victim flows. If there are no victim flows, contributor flows are allowed to send at high rates so that they can finish their transfers quickly to avoid creating victim flows.

The DCMS determines whether-or-not victim flows have been created by a congestion event by reading values of three types of switch port counters, namely, `PortXmitCongTime`, `PortXmitWait`, and `PortXmitData`. `PortXmitCongTime` is the amount of time a port has spent in a congested state. `PortXmitWait` indicates the amount of time a port has data to send but lacks flow-control credits. `PortXmitData` is the amount of data transmitted on the port (in 32-bit words). Specifically, DCMS uses the `perfquery` tool to gather values of above stated counters. Since `perfquery` uses General Management Packets (GMPs) that are subjected to flow control, DCMS uses a separate SL for these packets so that they are given higher priority than user-data packets, and not subject to congestion control.

The default settings for switch CC parameters are presented first, and then an algorithm for making dynamic modifications is described.

3.1. Default values for switch parameters

To begin with, we recommend that the CC `Threshold` in all switches be set to 15 [4], so that even at the slightest hint of congestion on one of its ports, a switch will start increasing the corresponding `PortXmitCongTime` counter, allowing the DCMS to react in a manner that prevents or mitigates the problem of cascading rate reductions.

We recommend setting the switch `Marking_Rate` to a high value so that few FECNs and corresponding BECNs are sent when a congestion event occurs. The fewer the BECNs, the smaller the injection rate reduction at the sending HCA. In other words, when `Marking_Rate` is high, contributor flows will continue to send data at high rates. The rationale is that the DCMS will determine whether a congestion event has created victim flows using the algorithm described in the next section, and if there are victim flows, the DCMS will reduce `Marking_Rate` to create more FECNs and BECNs, which in turn will cause contributor flows to throttle their injection rates. On the other hand, if the congestion event does not cause any victim flows, then the DCMS will allow the contributor flows to enjoy high throughput by not changing the `Marking_Rate` from its default high setting. The sooner a contributor flow ends, the lower the probability of it causing a victim flow.

3.2. Algorithm for dynamic modification of `Marking_Rate`

Algorithm 1 Dynamic Congestion Management

1: Read switch counters and compute $\Delta W(s,p,t), \Delta C(s,p,t), \Delta D(s,p,t), \forall s \in \mathbf{S},$ and $p \in \mathbf{P}_s$ in each sweep
2: Call Algorithm 2
3: Call Algorithm 3
4: Sleep until sweep timer τ expires; **go to** 1

The basic concept of the algorithm is as follows. If a congestion event causes a rate reduction in a neighboring switch port, the `Marking_Rate` parameter of the congested (root) port is lowered significantly so that the senders of contributor flows throttle their sending rates aggressively and congestion dissipates quickly. The DCMS monitors the affected neighboring switch port(s) to check if their link rates recovered after the lowering of the root-port `Marking_Rate`. If such a recovery is in evidence on even one affected neighboring switch port, the root-port `Marking_Rate` is kept low. A count is maintained for the number of inter-sweep intervals for

Algorithm 2 Check for newly congested ports and new victim ports

for each port (s, p) where $s \in \mathbf{S}$, $p \in \mathbf{P}_s$ **do**,
  if $(\Delta C(s,p,t) > T_C) \vee ((N(s,p) \in \mathbf{J}) \wedge (\Delta W(s,p,t) > T_W))$ **then** ▷ Is the port congested?
    for each port (r, q) where $(r \in \mathbf{S}, q \in \mathbf{P}_r, N(r,q) = (s,v), v \in \mathbf{P}_s)$ s.t. $(r,q) \notin \mathbf{V}(s,p)$ **do**
      if $\Delta W(r,q,t) > T_W$ **then** ▷ Is there a victim port?
        Add (r, q) to set $\mathbf{V}(s,p)$ ▷ Add port to set of victim ports for the congested port
        if $\mathbb{S}(s,p) \neq$`Low-MR` **then**
          ▷ Another victim port could have previously caused this state change
          $\mathbb{M}(s,p) \leftarrow$ `Low` ▷ A low setting will lead to a reduction in sending rates of congestion-causing flows
          $\mathbb{S}(s,p) =$`Low-MR`
          $\mathbb{I}(s,p) \leftarrow 1$ ▷ Interval count to limit maximum duration for low `Marking_Rate` setting
        end if
      end if
    end for
  end if
end for

Algorithm 3 Monitor ongoing CRR congestion events and restore default operation

for each port (s, p) for which $(|\mathbf{V}(s,p)| \neq 0) \wedge (\mathbb{I}(s,p) \neq 1)$ **do**
  Increment $\mathbb{I}(s,p)$ by 1
  if $(\mathbb{I}(s,p) \leq T_I)$ **then** ▷ Low-marking-rate maximum duration not yet reached
    for each port $(r, q) \in \mathbf{V}(s,p)$ **do**
      if $(\Delta D(r,q,t-\tau) - \Delta D(r,q,t)) > T_D$ **then**
        ▷ Link utilization dropped signaling absence or completion of victim flows
        Remove port (r, q) from $\mathbf{V}(s,p)$
      end if
    end for
  end if
  if $(|\mathbf{V}(s,p)| == 0) \vee (\mathbb{I}(s,p) == T_I)$ **then**
    ▷ No more victim ports, or Low-marking-rate duration threshold is reached
    $\mathbb{M}(s,p) \leftarrow$ `Default`
    $\mathbb{S}(s,p) =$`Default-MR`
  end if
end for

which the Marking_Rate is kept low, and a low-MR (Marking_Rate) duration threshold is used to limit the maximum number of intervals for which the congested-port Marking_Rate is kept low. Given the dynamic nature of flows, the DCMS cannot know for certain whether-or-not all victim flows have ended, and therefore it restores the root-port Marking_Rate to its high default value when the Low-MR duration threshold is crossed. If some of the victim flows are still ongoing, a second cycle of low Marking_Rate and congestion-event monitoring will be started. The cycle will be repeated multiple times if needed.

The DCMS procedure is described in pseudode in Algo. 1. Periodically, the DCMS reads the three types of switch-port counters described earlier. This operation is referred to as a "sweep." For simplicity, Line 1 states that the counters are read for all ports of all switches (see Table 1 for notation), but in practice, the DCMS can build up historical information on ports that experience congestion, and limit its reading of the PortXmitCongTime to just those ports that suffer from congestion. Line 1 of Algo. 1 shows that changes in PortXmitCongTime (ΔC), PortXmitWait (ΔW), and PortXmitData (ΔD) counters are computed by the DCMS. Algo. 1 then calls Algo. 2 and Algo. 3 as shown in the pseudocode. Algo. 1 is executed after each sweep, which is conducted at time intervals of τ, a configurable parameter.

Algo. 2 identifies congested ports, and then examines the PortXmitWait counter of ports in neighboring switches to determine if the congestion event is localized or a CRR event. Multiple ports in neighboring switches could suffer from a rate reduction due to a congestion event, i.e., there could be multiple victim ports for a single congestion event. Therefore a set $\mathbf{V}(s,p)$ is created to store the identifiers of the victim ports caused by congestion of root port p of switch s (henceforth the notation (s,p) will be used to denote this port). As soon as the first victim port is identified, the Marking_Rate $\mathbb{M}(s,p)$ is immediately set to Low, so that the senders of contributor flows decrease their injection rates rapidly. A state variable $\mathbb{S}(s,p)$ is used in the DCMS to track the Marking_Rate value set for the port in order to avoid sending unnecessary messages from the DCMS to a switch. An interval count $\mathbb{I}(s,p)$ tracks the number of inter-sweep intervals since the Marking_Rate of port (s,p) was set to Low. Details are provided in the pseudo-code of Algo. 2.

The purpose of Algo. 3 is to monitor counter values during ongoing CRR congestion events, and to restore the Marking_Rate of a congested port. The challenge lies in determining when to increase the Marking_Rate of a congested port back to its Default value. The key point is that a victim port may or may not have victim flows. A contributor flow, i.e., a flow that traverses the congested port, could make a neighboring switch port through which it passes a victim port because of link-by-link flow control as explained in Section 2. Therefore, a victim port does not necessarily need to have a victim flow. If there is no victim flow, the Marking_Rate of the congested port should be rapidly restored to its Default value. On the other hand, if there is a victim flow through a victim port, the Marking_Rate of the congested port should be held Low.

Since the DCMS does not have per-flow information, it makes conjectures about the presence or absence of victim flows through victim ports. Our solution is based on an observation that when the Marking_Rate of a congested port is set to Low, the contributor flows will suffer from a rapid rate reduction, which in turn will cause the neighboring victim port to return to its full-capacity state, allowing bulk-data victim flows to enjoy a rapid increase in throughput. Therefore, the DCMS observes the changes in the PortXmitData counter of victim ports for an ongoing CRR congestion event. An increase in the observed utilization of a victim port is assumed to indicate the presence of a victim flow, because if only contributor flows passed

through the victim port, dropping the `Marking_Rate` of the congested port to `Low` would cause a rate reduction in the contributor flows, and correspondingly in the utilization of the victim port. On the other hand, a decrease in the observed utilization of a victim port is interpreted by the DCMS as an absence or completion of victim flows passing through the victim port.

The DCMS will reset the `Marking_Rate` of congested ports back to the high `Default` value if there are no more victim ports or if the duration for which a port's `Marking_Rate` was held low exceeds a threshold T_I, which is the Low-MR duration threshold (see Table 1). Details are provided in the pseudo-code of Algo. 3.

In summary, the default setting of switch `Marking_Rate` is chosen to be a high value so that if a congestion event is localized, the contributor flows are allowed to enjoy high throughput so that they end quickly. If, on the other-hand, the congestion causes CRRs in neighboring switch port rates, then the DCMS reduces the `Marking_Rate` of the congested (root) port to reduce the impact of contributor flows on victim flows. The DCMS then plays a guessing game of whether-or-not victim flows are passing through victim ports, when victim flows end, when ports in neighboring switches stop being victim ports, and when a congestion event ends. The DCMS needs to trade off the negative impact of a congestion event on victim flows, while simultaneously ensuring that contributor flows are provided with the opportunity to send data at high rates and thus end quickly.

A limitation of this solution is sweep overhead, as was noted in the dFtree solution [5]. To reduce sweep overhead, the DCMS could rely on historical data to limit its reading of switch counters to just those ports that typically suffer from congestion (e.g., ports connected to disk subsystems). Further the DCMS could be configured to use two inter-sweep intervals, a longer interval until a congestion event is detected, and a shorter interval while a congestion is in progress. A short interval is required for an effective assessment by the DCMS about whether-or-not victim flows exist after a `Marking_Rate` reduction. The initial longer interval would most likely cause the DCMS to miss short-duration congestion events. But we reason that if a congestion event is of short duration, its impact on other flows is necessarily limited, which could justify no DCMS action. On the other hand, long-lasting contributor flows should be throttled as they have more time in which to adversely affect other flows, and therein lies the value of DCMS.

4. Experiments

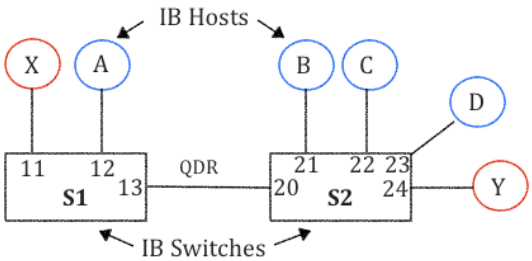


Figure 2. Experimental testbed; All HCA links are SDR

Two experiments were conducted to study CRR congestion events. Experiment I was used to study the impact of the default setting of `Marking_Rate`. In Experiment II, our DCMS prototype was executed to dynamically modify the `Marking_Rate`, and results were collected to study the impact of low-MR duration threshold in the DCMS solution.

Section 4.1 describes the testbed setup used for the experiments. Section 4.2 describes the execution and results of Experiment I. Section 4.3 describes the execution and evaluation of Experiment II.

4.1. Experimental configuration

The experimental setup consists of 6 hosts interconnected by two InfiniBand switches as shown in Fig. 2. Each host has an Intel Xeon E5/Core i7 CPU, 64 GB RAM, PCIe-2 bus, and a Mellanox MT4099 HCA. All hosts run OFA OFED-3.5-2 on top on Scientific Linux v6. In all experiments, OpenSM v3.3.18 runs on one of the hosts.

Each node's HCA was configured to operate at 4xSDR rate (approx. 8 Gbps). The inter-switch link rate was set to 4xQDR rate (approx. 32 Gbps). For simplicity, we refer to 4xSDR as SDR and 4xQDR as QDR in the rest of the paper.

A custom software program called `blast` was used to create high-throughput, memory-to-memory transfers. Each flow sent 140 messages, each of size 128 MiB.

4.2. Experiment I: Default setting

The goal of this experiment was to determine a default value for switch `Marking_Rate`. Since reassembling packets into flows to identify senders of contributor flows inside the network is a compute-intensive operation, the DCMS does not know the senders of contributor flows and hence cannot modify the HCA CC parameters. Nevertheless, the values of these HCA CC parameters will influence congestion-recovery time. Therefore we study the impact of `CCTI_Timer`.

Our main finding is that for the testbed used, if `CCTI_Timer` is set to values in the 75-300 range at the HCAs, then a default value of 64 or 128 for switch `Marking_Rate` is sufficient to allow contributor flows to enjoy high throughput if the congestion event is localized.

The `Marking_Rate` was varied from 0 to 2048, and the `CCTI_Timer` was set to 75, 150, and 300. The other CC parameters were set as follows (i) At each HCA: `CCTI_Increase`: 1, `CCTI_Limit`: 0, and (ii) At each switch: `Threshold`: 15.

`Blast` flows were started sequentially as follows: (i) X-Y flow, (ii) B-D flow at 8s, (iii) C-D flow at 13s, and (iv) A-D flow at 28s (See Fig. 2). Ideally, the X-Y flow should not be affected by the other flows, and the remaining flows should each receive a one-third share of the SDR rate of port (S2, 23). The (s, p) notation is used to identify port p of switch s, and the reader is referred to Fig. 2 for switch and port numbers. The X-Y flow did enjoy throughput close to SDR (which was the rate of the HCAs) under some values of `Marking_Rate` but not others, as seen in our discussion of the results below.

Fig. 3 shows graphs corresponding to three settings of the `CCTI_Timer` and `Marking_Rate`: (i) 75 and 0, respectively, (ii) 300 and 0, respectively, and (iii) 75 and 2048, respectively. We use the shorthand notation 75-0, 300-0, and 75-2048 for these three cases. In the first two cases, where `Marking_Rate` was 0, the X-Y flow enjoyed unhindered SDR throughput; however, in the third case, the X-Y flow was limited to one-third SDR. This finding is explained below.

Fig. 3a shows that the sum total of the throughput values for the B-D, C-D and A-D flows was only 3.1 Gbps from time 40.32 s to 301.8 s in the 75-0 case. This number is even lower at 1.98 Gbps for the 300-0 case as seen in Fig. 3b. At the higher value (300) of the `CCTI_Timer`, the CCTI of flows will be decreased at a lower rate, and therefore inter-packet injection times stay high leading to a lower sending rate. This behavior explains why the total throughput for

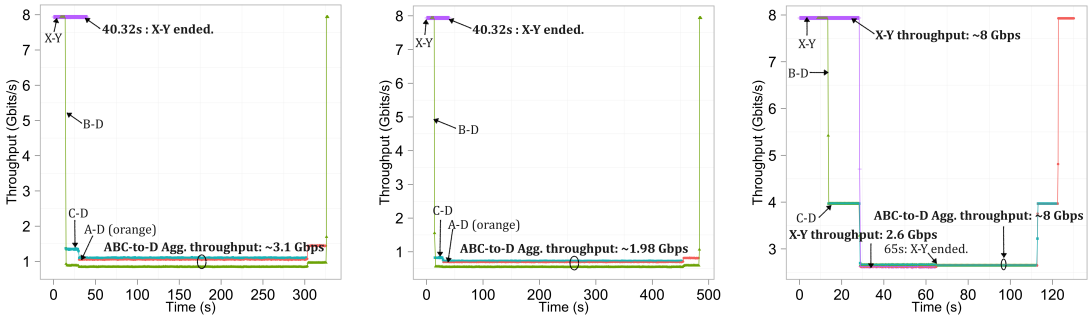
the three flows is lower in the 300-0 case. In these two cases, 75-0 and 300-0, the X-to-Y flow ran unhindered at 7.9 Gbps (close to SDR), and completed its transfer in 40.32 s. Since the aggregate throughput of the B-D, C-D and A-D flows was less than the 8 Gbps rate of port (S2, 23), the congestion dissipated quickly. Since the inter-switch link was QDR, the sum total throughput of the X-Y and A-D flows was less than the inter-switch link rate. No congestion was recorded in the `PortXmitCongTime` counter of port (S1, 15). This is because the sum total of the input-flow rates destined for port (S1, 15) was less than the QDR rate of this port.

In the third case, 75-2048, only 1 in 2048 packets were marked by switch S2 in the contributor flows (A-D, B-D, C-D). This results in a low BECN arrival rate at the senders, A, B and C, causing these senders to maintain fairly high sending rates. Therefore, the sum total throughput of the B-D, C-D, and A-D flows was close to the SDR theoretical maximum rate of 8 Gbps, as seen in Fig. 3c. On the other hand, the X-Y flow was impacted when the A-D flow was initiated at 28 s. The X-Y flow throughput dropped from 7.9 Gbps to 2.6 Gbps at time 29.5 s, as seen in Fig. 3c, and stayed at this rate until the X-Y flow ended at 65 s. In this case, the X-Y flow is a victim flow.

An explanation of why the X-Y flow received only 2.6 Gbps (one-third SDR) lies in the rate-reduction of port (S1, 15) caused due to a lack of flow-control credits from port (S2, 20) (see Fig. 2). Unlike in the 75-0 and 300-0 cases, when the `PortXmitWait` counter at port (S1, 15) was 0, in the 75-2048 case, the `PortXmitWait` counter recorded 2.5×10^9 ticks at the end of the flows. As a tick was 22ns in the testbed network, and the duration of the test was 130 s, the transmitter was not allowed to send data for 55 s out of the 130 s duration, which means that the effective link rate was lowered to 13.54 Gbps from the original 32 Gbps QDR link capacity. Hence this rate reduction at port (S1, 15) would have caused the input-side buffers at ports 11 and 12 to fill up, causing these ports to send FCPs with limited credits to the HCAs at X and A, respectively. The `PortXmitWait` at HCAs X and A also built up to 1.9×10^8 ticks and 4.8×10^8 ticks, respectively. The packet scheduler at port (S1, 15) would have served packets of the X-Y and A-D flows from ports 11 and 12, respectively, in round-robin mode, and therefore both these flows get the same 2.6 Gbps throughput. To determine a suitable default setting, we repeated the experiment for other settings of `Marking_Rate` besides 0 and 2048. Fig. 4 shows the results for `Marking_Rate` values in powers of 2, i.e., 8, 16, $\cdots$, 128. As seen earlier, when the marking rate was 0, the X-Y flow duration was 40.32 sec under all three settings of the `CCTI_Timer` (75, 150, and 300), which is the same duration as that of an unhindered X-Y flow. When `Marking_Rate` was 0, the `PortXmitCongTime` of (S2, 23) reached only 330K, 216K, and 130K, for the three values of `CCTI_Timer`, 75, 150, and 300, respectively. However, for other values of `Marking_Rate`, starting from 8 to 128, the `PortXmitCongTime` of (S2, 23) reached approximately 850K. In other words, when `Marking_Rate` is set to 0, the congestion dissipates faster. The completion time of the X-Y flow was close to 65 for higher `Marking_Rate` values, reaching this level sooner for the smaller 75 setting of the `CCTI_Timer` as seen in Fig. 4a.

Fig. 4b shows that when the marking rate was 0, the total throughput of the A-D, B-D, C-D flows added up to only 3.18, 2.46, and 2.1 Gbps with `CCTI_Timer` values of 75, 150 and 300, respectively, all of which are well below the SDR rate. This illustrates that a marking rate of 0 is needed to avoid the effects of congestion on victim flows, but that this advantage is achieved by sacrificing throughput of contributor flows. With a `Marking_Rate` of 64, the aggregate throughput of the three contributor flows was 8 Gbps for all three values of `CCTI_Timer` at a cost to the victim flow.

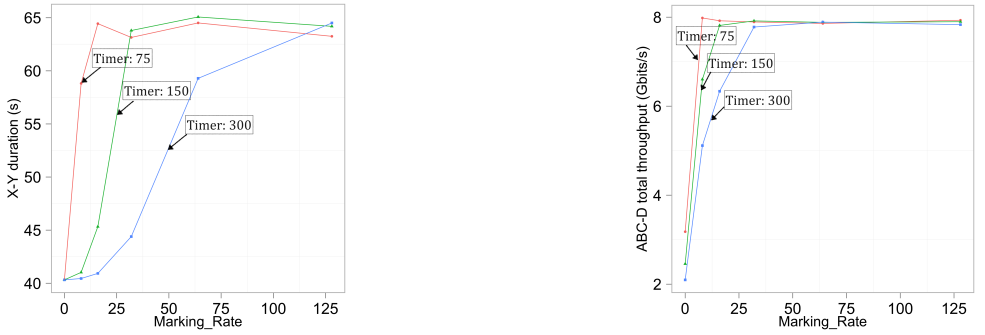
In summary, this experiment shows us that a default value of 64 or 128 can be used for switch `Marking_Rate` if HCAs `CCTI_Timer` values lie in the range 75-300. Further, it showed that the `Low` value of the `Marking_Rate` used by DCMS should be 0.



(a) 75-0 (b) 300-0 (c) 75-2048

Figure 3. Per-flow throughput as a function of time; `CCTI_Timer-Marking_Rate` is shown below each graph

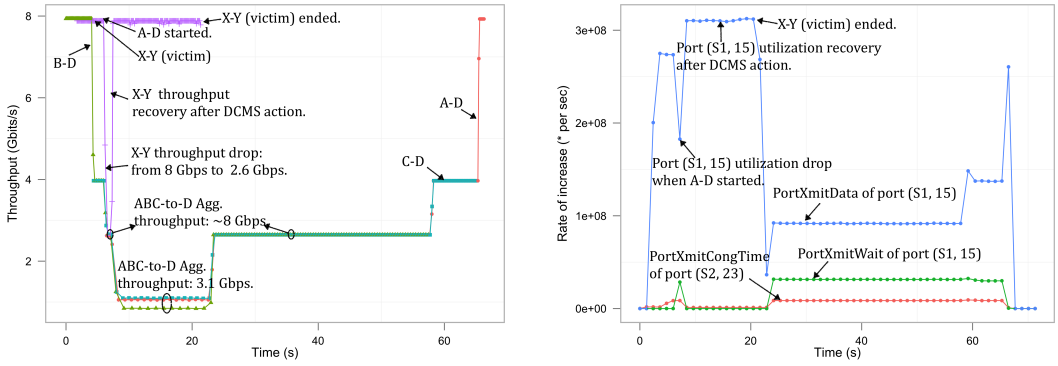
4.3. Experiment II: DCMS



(a) X-Y victim flow duration (b) Aggregate throughput of contributor flows

Figure 4. Illustrates tradeoff between victim-flow and contributor-flow performance

In this experiment, our DCMS prototype was executed to dynamically modify the `Marking_Rate`, and the results were collected to study the impact of low-MR duration threshold in the DCMS solution. Flows were started sequentially in the following order: B-D, X-Y at 2s,



(a) Flow throughput vs. time (b) Port counter values

Figure 5. Scenario 1: Longer Low-MR duration threshold (12 sec)

C-D at 5s, and A-D at 7s. The `CCTI_Timer` was set to 75, and a default value of 128 was used for the `Marking_Rate`. The thresholds T_W and T_C were set to 27.4M and 8M, respectively. Two scenarios were created by varying the Low-MR duration threshold. In the first scenario, the victim flow ended before the threshold was reached, while in the second scenario, the contributor and victim flows continued past the threshold, and therefore, the second reduction in the `Marking_Rate` followed by congestion-event monitoring was required.

Fig. 5a shows the results for Scenario 1. The B-D flow enjoyed 8 Gbps when it started, as did the X-Y flow when it started. At 4.5s, when the C-D flow was started, the B-D and C-D flows each got roughly 4 Gbps, while the X-Y flow got 8 Gbps. As the aggregate rate of the B-D and C-D flows exceeded (S2, 23) port capacity, the rate of increase of `PortXmitCongTime` of port (S2, 23) increased at 4.8s from small values to 6.0×10^6 ticks/s, as seen in Fig. 5b. However, `PortXmitWait` of port (S1, 15) remained unchanged (i.e., rate of increase, as seen in Fig. 5b, was 0) as the A-D flow had not yet started. Meanwhile, `PortXmitData` of port (S1, 15) registered a growth rate of 2.7×10^8 words/s as seen in Fig. 5b, which means X-Y flow was sending packets at a high rate. Despite the increase in `PortXmitCongTime` of port (S2, 23), the controller took no action as the congestion was localized, i.e., there were no victim ports. In other words, the condition of Line 4 in Algo. 2 was not met.

When the A-D flow was started, the throughput of each flow destined to D was one-third of the port (S2, 23) rate (roughly 2.67 Gbps) as seen in Fig. 5a. In addition, the throughput of the X-Y flow also dropped from 8 Gbps to 2.67 Gbps. Fig. 5b shows that the `PortXmitWait` of port (S1, 15) started increasing at 2.8×10^7 ticks/s. Simultaneously, the growth-rate of `PortXmitData` of port (S1, 15) dropped from 2.7×10^8 words/s to 1.8×10^8 words/s. Together, these port counters illustrate the effect of rate reduction at port (S1, 15) caused by the congestion at port (S2, 23). At 7s, the T_C threshold and the T_W threshold were crossed (Lines 2 and 4 of Algo. 2), which caused the DCMS to lower `Marking_Rate` of port (S2, 23) to 0 (Line 8 of Algo. 2). The crossing of both thresholds was an indication that congestion at port (S2, 23) created a victim port (S1, 15).

At 8.5s, as a result of setting `Marking_Rate` to `Low`, the sending rates of flows B-D, C-D, and A-D decreased causing their throughput to drop from 2.67 Gbps to 1.23 Gbps. This action relieved congestion, and the throughput of the victim X-Y flow rebounded from 2.67 to roughly 8 Gbps as seen in Fig. 5a. Fig. 5b shows that at this point, the growth-rate of `PortXmitData` of port (S1, 15) grew to 3.1×10^8 words/s, while the growth rates of `PortXmitCongTime` at port (S2, 23) and `PortXmitWait` at port (S1, 15) dropped back down. From 8.5s to 23s, the controller did not reset the port (S2, 23) `Marking_Rate` to `Default` because the Low-MR duration threshold (12 s) was not crossed. During this time interval, the condition of Line 5 of Algo. 3 was not met, and therefore the `Marking_Rate` of port (S2, 23) was held at 0. This allowed the X-Y flow to complete at 22 s as seen in Fig. 5a.

Soon after, when the Low-MR duration threshold was crossed, the DCMS reconfigured the `Marking_Rate` of port (S2, 23) to 128 (its default value), which allowed the contributor flows to recover. As a result, each of A-D, B-D, C-D flows recovered throughput from 1.23 Gbps to 2.67 Gbps (aggregate of 8 Gbps shown in Fig. 5a). The growth rate of `PortXmitData` of port (S1, 15) remained at 0.91×10^8 words/s until B-D flow ended, at which point the rate of the A-D flow increased, causing the growth rate of `PortXmitData` of port (S1, 15) to increase to 1.37×10^8 words/s. When C-D ended and the A-D flow started enjoying 8 Gbps as seen in Fig. 5a, the growth rate of `PortXmitData` of port (S1, 15) increased to 2.6×10^8 words/s as seen in Fig. 5b.

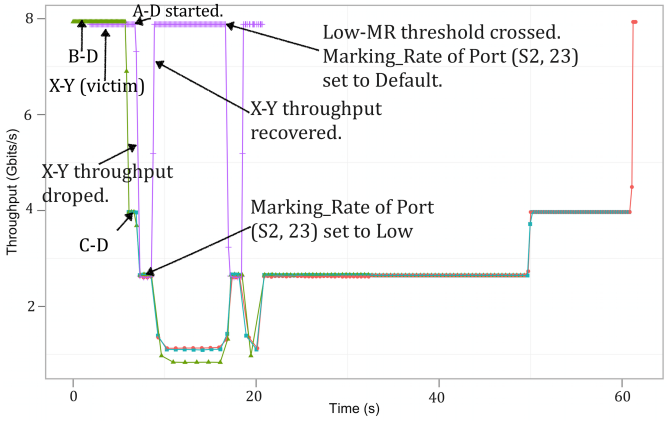


Figure 6. Scenario 2: Shorter Low-MR duration threshold (6 sec)

Finally, when A-D ended and there were no more flows passing through port (S1, 15), the growth rate of PortXmitData dropped back to 0.

Fig. 6 shows Scenario 2 results in which the Low-MR duration threshold was set to 6 sec, which caused the DCMS to reset the `Marking_Rate` of port (S2, 23) to its default value at 17 sec. This operation caused the X-Y victim flow to suffer another drop in its throughput. The DCMS observed a similar build-up of the port counters resulting in a second reduction in the `Marking_Rate` of port (S2, 23), which allowed the X-Y victim flow to recover its throughput. This example illustrates that the DCMS can protect victim flows from the effects of congestion even with no knowledge of flows.

In summary, the above experiments demonstrated the feasibility of deploying a DCMS to manage CC parameters in a dynamic manner such that both victim flows and contributor flows can be served effectively.

5. Related Work

Prior work on InfiniBand congestion control includes simulation and experimental studies [1, 3, 6, 7], recommendations for setting CC parameters [4, 8], and new methods to combat the effects of congestion [9–14].

Our work builds on the above-cited literature by extending our understanding of the effects of link-by-link flow control on congestion in InfiniBand networks. While terms such as congestion spreading [8], and forests of congestion trees [1], capture the effects of link-by-link flow control, in Section 2, we offered a new term cascading rate-reductions to describe the idea that as such links that are behind a congestion point do not themselves suffer from congestion, but rather a reduction in rate as explained in Section 4.2.

With regards to setting CC parameters, our contribution is to determine default values for switch `Marking_Rate`, a parameter that was not considered in the work by Pfister, et al. [8]. The work by Gran, et al. [6] considered switch `Marking_Rate` and HCA `CCTI_Timer` as we did, but stated that the question of how to set CC parameters was a "subject of ongoing research." Our contribution to this subject, which is a dynamic modification of `Marking_Rate`, is a new advance. An adaptive marking rate solution was patented [15], but it requires knowledge about flows.

Of the work on new methods to combat the effects of congestion, Regional Explicit Congestion Notification (RECN) [9, 11] and Destination-Based Buffer Management [12] are effective

but require modifications of the switches as they redirect contributor flow packets to separate queues. Our objective is to improve congestion management in deployed InfiniBand networks and hence a design goal was to require no modifications to InfiniBand switches.

The VOQsw methodology [10], vFtree [13], Flow2SL [14] have the same objective of offering a solution that does not require switch modifications, and leverage InfiniBand's service lane (SL) and virtual lane (VL) features. Our DCMS solution is complementary to these methodologies as it would handle the intra-VL hogging problem.

Our work is the most similar to the dFtree solution proposed by Guay et al. [5] in that the dFtree solution also uses performance counters. However, our congestion recovery uses modifications to switch `Marking_Rate`, while the dFtree solution reassigns hot flows to a slow virtual lane. This paper was written in 2011, and states that since congestion control was newly introduced, it was not available in all switches and HCAs, and therefore, the dFtree solution was designed to work without congestion control.

Conclusions

This work has demonstrated the feasibility of dynamically modifying a switch congestion-control parameter called `Marking_Rate` to enable flows victimized by a congestion event to recover their throughput rapidly. We conclude that even without per-flow information, it is feasible for a Dynamic Congestion Management System (DCMS), which is software running on an external server, to use just the information in switch port counters to make educated guesses about the presence or absence of victim flows during a congestion event. Our experimental work has demonstrated that if the `Marking_Rate` is kept too low, flows that cause congestion would experience severe rate reductions, which would prolong the congestion-event duration and correspondingly increase the probability of creating victim flows. But without a dynamic management system, `Marking_Rate` cannot be kept high because if a congestion event occurs, such a setting could create many victim flows. Thus, the value and feasibility of a dynamic congestion management system has been demonstrated in this work.

Acknowledgments

This work was supported by NSF grants CNS-1116081, OCI-1127340, ACI-1340910, CNS-1405171, ACI-0958998, OCI-1127228, and OCI-1127341. The authors also thank Patrick MacArthur, David Wyman, and Chuck Valenza, University of New Hampshire, for testbed support.

References

1. Gran EG, Reinemo SA, Lysne O, Skeie T, Zahavi E, Shainer G. Exploring the Scope of the InfiniBand Congestion Control Mechanism. In: Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International; 2012. p. 1131–1143.

2. InfiniBand Trade Association. InfiniBand Architecture Specification Volume 1, Release 1.3; 2015. Available from: `http://infinibandta.org`.

3. Gran EG, Reinemo SA. InfiniBand Congestion Control: Modelling and Validation. In: Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques.

SIMUTools '11. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering); 2011. p. 390–397.

4. Gran EG, Eimot M, Reinemo SA, Skeie T, Lysne O, Huse LP, et al. First experiences with congestion control in InfiniBand hardware. In: Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on; 2010. p. 1–12.

5. Guay WL, Reinemo SA, Lysne O, Skeie T. dFtree: A Fat-tree Routing Algorithm Using Dynamic Allocation of Virtual Lanes to Alleviate Congestion in InfiniBand Networks. In: Proceedings of the First International Workshop on Network-aware Data Management. NDM '11. New York, NY, USA: ACM; 2011. p. 1–10.

6. Gran EG, Zahavi E, Reinemo SA, Skeie T, Shainer G, Lysne O. On the Relation between Congestion Control, Switch Arbitration and Fairness. In: Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on; 2011. p. 342–351.

7. Santos JR, Turner Y, Janakiraman G. End-to-end congestion control for InfiniBand. In: INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies. vol. 2; 2003. p. 1123–1133 vol.2.

8. Pfister G, Gusat M, Denzel W, Craddock D, Ni N, Rooney W, et al. Solving hot spot contention using InfiniBand architecture congestion control. In: Proceedings In High Performance Interconnects for Distributed Computing, Research Triangle Park, NC; 2005. .

9. Duato J, Johnson I, Flich J, Naven F, Garcia P, Nachiondo T. A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks. In: High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on. IEEE; 2005. p. 108–119.

10. Gomez ME, Flich J, Robles A, Lopez P, Duato J. VOQSW: a methodology to reduce HOL blocking in InfiniBand networks. In: Parallel and Distributed Processing Symposium, 2003. Proceedings. International; 2003. p. 10 pp.–.

11. Garcia PJ, Quiles FJ, Flich J, Duato J, Johnson I, Naven F. Efficient, Scalable Congestion Management for Interconnection Networks. Micro, IEEE. 2006 Sept;26(5):52–66.

12. Nachiondo T, Flich J, Duato J. Buffer Management Strategies to Reduce HoL Blocking. Parallel and Distributed Systems, IEEE Transactions on. 2010 June;21(6):739–753.

13. Guay WL, Bogdanski B, Reinemo SA, Lysne O, Skeie T. vFtree - A Fat-Tree Routing Algorithm Using Virtual Lanes to Alleviate Congestion. In: Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International; 2011. p. 197–208.

14. Escudero-Sahuquillo J, Garcia PJ, Quiles FJ, Reinemo SA, Skeie T, Lysne O, et al. A new proposal to deal with congestion in InfiniBand-based fat-trees. Journal of Parallel and Distributed Computing. 2014;74(1):1802 – 1819.

15. Zahavi E. InfiniBand adaptive congestion control adaptive marking rate. Google Patents; 2010. US Patent App. 12/245,814.

DOI: 10.14529/jsfi160202

Many-Core Approaches to Combinatorial Problems: case of the Langford problem

M. Krajecki[1], *J. Loiseau*[1], *F. Alin*[1], *C. Jaillet*[1]

As observed from the last TOP500[2] list - November 2015 -, GPUs-accelerated clusters emerge as clear evidence. But exploiting such architectures for combinatorial problem resolution remains a challenge.
In this context, this paper focuses on the resolution of an academic combinatorial problem, known as the Langford pairing problem, which can be solved using several approaches. We first focus on a general solving scheme based on CSP (*Constraint Satisfaction Problem*) formalism and backtrack called the Miller algorithm. This method enables us to compute instances up to $L(2, 21)$ using both CPU and GPU computational power with load balancing.

As dedicated algorithms may still have better computation efficiency we took advantage of Godfrey's algebraic method to solve the Langford problem and implemented it using our multiGPU approach. This allowed us to recompute the last open instances, $L(2, 27)$ and $L(2, 28)$, respectively in less than 2 days and 23 days using best-effort computation on the ROMEO[3] supercomputer with up to 500,000 GPU cores.

Keywords: Combinatorial problems, parallel algorithm, GPU accelerators, CUDA, Langford problem.

Introduction

For many years now, GPUs usage has increased in the field of High Performance Computing. The TOP500 list of the world's most powerful supercomputers contains more than 52 systems powered by NVIDIA Kepler GPUs. In the latest list a number of hybrid machines increased compared fourfold with the previous list.
Since 2007, NVIDIA has offered a general GPUs programming interface: *Compute Unified Device Architecture* (CUDA). This study is based on this physical and logical architecture which requires massively parallel programming and a new vision for the implementation of resolution algorithms.

The Langford pairing problem is a very irregular combinatorial problem and thus is a bad candidate for GPU computation which requires vectorized and regularized tasks. Hopefully there are many ways to regularize the computation in order to take advantage of the multiGPU cluster architectures.

This paper is structured as follows: we first present the background with the Langford problem and multiGPU cluster. The next section describes our method concerning the Miller algorithm on such architectures. Then we expose our multiGPU solution to solve the Langford problem based on the Godfrey algorithm. Finally, we present some concluding remarks and perspectives.

[1]University of Reims Champagne-Ardenne
[2]http://www.top500.org
[3]https://romeo.univ-reims.fr/pages/aboutUs

Table 1. Solutions and time with differents methods

Instance	Solutions	Method	Computation time
L(2,3)	1	Miller algorithm	-
L(2,4)	1		-
...	...		...
L(2,16)	326,721,800		120 hours
L(2,19)	256,814,891,280		2.5 years (1999) DEC Alpha
L(2,20)	2,636,337,861,200	Godfrey algorithm	1 week
L(2,23)	3,799,455,942,515,488		4 days with CONFIIT
L(2,24)	46,845,158,056,515,936		3 months with CONFIIT
L(2,27)	111,683,611,098,764,903,232		-
L(2,28)	1,607,383,260,609,382,393,152		-

1. Background

1.1. Langford problem

C. Dudley Langford gave his name to a classic permutation problem [1, 2]. While observing his son manipulating blocks of different colors, he noticed that it was possible to arrange three pairs of different colored blocks (yellow, red, blue) in such a way that only one block separates the red pair - noted as pair 1 - , two blocks separate the blue pair - noted as pair 2 - and finally three blocks separate the yellow one - noted as pair 3 - , see Fig. 1.

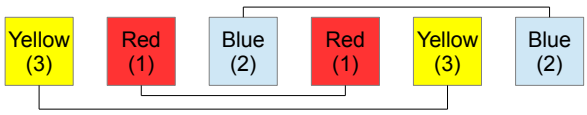


Figure 1. L(2,3) arrangement

This problem has been generalized to any number n of colors and any number s of blocks having the same color. $L(s, n)$ consists in searching for the number of solutions to the Langford problem, up to a symmetry. In November 1967, Martin Gardner presented $L(2, 4)$ (two cubes and four colors) as being part of a collection of small mathematical games and he stated that $L(2, n)$ has solutions for all n such that $n = 4k$ or $n = 4k - 1$ ($k \in \mathbb{N} \setminus \{0\}$). The central resolution method consists in placing the pairs of cubes, one after the other, on free places and backtracking if no place is available (see Fig. 3 for a detailed algorithm).

The Langford problem has been approached in different ways: discrete mathematics results, specific algorithms, specific encoding, constraint satisfaction problem (CSP), inclusion-exclusion ... [3–6]. In 2004, the last solved instance, $L(2, 24)$, was computed by our team using a specific algorithm. (see Table 1); $L(2, 27)$ and $L(2, 28)$ have just been computed but no details were given.

The main efficient known algorithms are the following: the Miller backtrack method, the Godfrey algebraic method and the Larsen inclusion-exclusion method. The Miller technique is based on backtracking and can be modeled as a CSP; it allowed us to move the limit of explicits solutions building up to $L(2, 21)$ but combinatorial explosion did not allow us to go further.

Then, we use the Godfrey method to achieve $L(2,24)$ more quickly and then recompute $L(2,27)$ and $L(2,28)$, presently known as the last instances. The Larsen method is based on inclusion-exclusion [6]; although this method is effective, practically the Godfrey technique is better. The lastest known work on the Langford Problem is a GPU implementation proposed in [7] in 2015. Unfortunately this study does not provide any performance considerations but just gives the number of solutions of $L(2,27)$ and $L(2,28)$.

1.2. MultiGPU clusters and the ROMEO supercomputer

GPUs always come with CPUs which delegate them part of their computation. Let us consider a cluster as a set of one CPU and one or more GPU(s), which we call *machines*. We see these clusters as 3-level parallelism structures (as described in 2.1.4), with communications between nodes and/or machines, CPUs that prepare computation and finally delegate part of it to the GPUs. When the problem can be split into a finite number of independent tasks, it is possible to distribute them over the machines. That permits to make an efficient use of the cluster hardware. Depending on the way of computation submission we can use either a static multinode reservation with one job including MPI client-server tasks distribution, or a best-effort dynamic reservation using several one-node jobs for independent tasks.

As the execution model of GPUs is based on SIMT (Single Instruction Multiple Threads), the same instruction flow is shared by all the threads that execute synchronously by *warp* teams [8, 9]. The divergences in this flow are handled by the NVIDIA GPUs scheduler, but lead to synchronization between threads and an efficiency loss. This is the reason why we intend to provide regular resolution algorithms for an efficient use of the GPU capabilities and, moreover, with multiGPU clusters.

ROMEO supercomputer - All the tests below were led on the ROMEO cluster available at the University of Reims Champagne-Ardenne (France). It provides 130 nodes, each composed of 2 Ivy Bridge CPUs (8 cores), 2.6GHz and 2 Tesla K20Xm GPUs.

We use the nodes as two independent machines with one eight core CPU and one GPU attached, linked by PCIe-v3. This allows having 260 machines for computation, each containing 32GB RAM memory. A K20Xm GPU has 6GB memory, 250GB/s of bandwidth, 2688 CUDA cores including 896 double precision cores.

2. Miller algorithm

In this part we present our multiGPU cluster implementation of the Miller's algorithm. First, we introduce the backtrack method. Then we present our implementation in order to fit the GPUs architecture. The last section presents our results.

2.1. Backtrack resolution

As presented above, the Langford problem is known to be a highly irregular combinatorial problem. We first present here the general tree representation and the ways we regularize the computation for GPUs. Then we show how to parallelize the resolution over a multiGPU cluster.

2.1.1. Langford's problem tree representation

In [10], we propose to formalize the Langford problem as a CSP (*Constraint Satisfaction Problem*), first introduced by Montanari in [11], and show that an efficient parallel resolution is possible. CSP formalized problems can be transformed into tree evaluations. In order to solve $L(2, n)$, we consider the following tree of height n: see example of $L(2, 3)$ in Fig. 2.

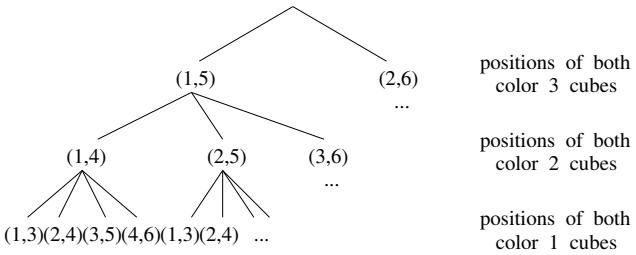


Figure 2. Search tree for $L(2, 3)$

- Every level of the tree corresponds to a color.
- Each node of the tree corresponds to the placement of a pair of cubes without worrying about the other colors. Color p is represented at depth $n - p + 1$, where the first node corresponds to the first possible placement (positions 1 and $p+2$) and i^{th} node corresponds to the placement of the first cube of color p in position i, $i \in [1,\ 2n - 1 - p]$.
- Solutions are leaves generated without any placement conflict.

There are many ways to browse the tree and find the solutions: *backtracking*, *forward-checking*, *backjumping*, etc [12]. We limit our study to the naive *backtrack* resolution and choose to evaluate the variables and their values in a static order; in a depth-first manner, the solutions are built incrementally and if a partial assignment can be aborted, the branch is cut. A solution is found each time a leaf is reached.

The recommendation for performance on GPU accelerators is to use non test-based programs. Due to its irregularity, the basic *backtracking* algorithm, presented on Fig. 3, is not supposed to suit the GPU architecture. Thus a vectorized version is given when evaluating the assignments at the leaves' level, with one of the two following ways: assignments can be prepared on each tree node or totally set on final leaves before testing the satisfiability of the built solution (Fig. 4).

```
while not done do
 test pair           <- test
 if successful then
   if max depth then
     count solution
     higher pair
   else
     lower pair      <- remove
 else
   higher pair       <- add
```

Figure 3. Backtrack algorithm

```
for pair 1 positions
  assignment                  <- add
  for pair 2 positions
    assignment                <- add
    for ...
      for pair n positions
        assignment            <- add
        if final test ok then
          count solution
```

Figure 4. Regularized algorithm

2.1.2. Data representation

In order to count every Langford problem solution, we first identify all possible combinations for one color without worrying about the other ones. Each possible combination is coded within an interger, one bit to 1 corresponding to a cube presence, a 0 to its absence. This is what we called a *mask*. This way Fig. 5 presents the possible combinations to place the one, two and three weight cubes for the $L(2,3)$ Langford instance.

Furthermore, the masks can be used to evaluate the partial placements of a chosen set of colors: all the 1s correspond to occupied positions; the assignment is consistent *iff* there are as many 1s as the number of cubes set for the assignment.

With the aim to find solutions, we just have to go all over the tree and *sum* one combination of each of the colors: a solution is found *iff* all the bits of the sum are set to 1.

Each route on the tree can be evaluated individually and independently; then it can be evaluated as a thread on the GPU. This way the problem is massively parallel and can be, indeed, computed on GPU. Fig. 6 represents the tree masks' representation.

	pair 1	pair 2	pair 3
1	0 0 0 1 0 1	0 0 1 0 0 1	0 1 0 0 0 1
2	0 0 1 0 1 0	0 1 0 0 1 0	1 0 0 0 1 0
3	0 1 0 1 0 0	1 0 0 1 0 0	
4	1 0 1 0 0 0		

Figure 5. Bitwise representation of pairs positions in $L(2,3)$

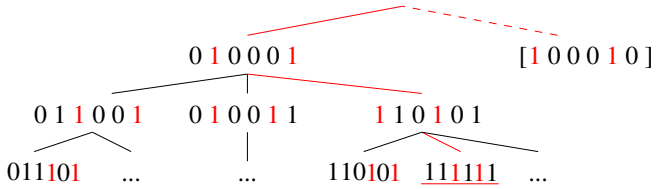


Figure 6. Bitwise representation of the Langford $L(2,3)$ placement tree

2.1.3. Specific operations and algorithms

Three main operations are required in order to perform the tree search. The first one, used for both backtrack and regularized methods, aims to add a pair to a given assignment. The second one, allowing to check if a pair can be added to a given partial assignment, is only necessary for the original backtrack scheme. The last one is used for testing if a global assignment is an available solution: it is involved in the regularized version of the Miller algorithm.

Add a pair - Top of Fig. 7 presents the way to add a pair to a given assignment. With a *binary or*, the new mask contains the combination of the original mask and of the added pair. This operation can be performed even if the position is not available for the pair (however the resulting mask is inconsistent).

Test a pair position - On the bottom part of the same figure, we test the positioning of a pair on a given mask. For this, it is necessary to perform a *binary and* between the mask and the pair.

$= 0$: *success*, the pair can be placed here

$\neq 0$: *error*, try another position

Final validity test - The last operation is for *a posteriori* checking. For example the mask 101111, corresponding to a leaf of the tree, is inconsistent and should not be counted among the solutions. The final placement mask corresponds to a solution *iff* all the places are occupied, which can be tested as $\neg mask = 0$.

Using this data representation, we implemented both *backtrack* and *regularized* versions of the Miller algorithm, as presented in Fig. 3 and 4.
The next section presents the way we hybridize these two schemes in order to get an efficient parallel implementation of the Miller algorithm.

2.1.4. Hybrid parallel implementation

This part presents our methodology to implement Miller's method on a multiGPU cluster.

Tasks generation - In order to parallelize the resolution we have to generate tasks. Considering the tree representation, we construct tasks by fixing the different values of a first set of variables [pairs] up to a given level. Choosing the development level allows to generate as many tasks as necessary. This leads to a *Finite number of Irregular and Independent Tasks* (*FIIT* applications [13]).

Cluster parallelization - The generated tasks are independent and we spread them in a client-server manner: a server generates them and makes them available for clients. As we consider the cluster as a set of CPU-GPU(s) machines, the clients are these machines. At the machine level, the role of the CPU is, first, to generate work for the GPU(s): it has to generate sub-tasks, by continuing the tree development as if it were a second-level server, and the GPU(s) can be considered as a second-level client(s).
The sub-tasks generation, at the CPU level, can be made in parallel by the CPU cores. Depending on the GPUs number and their computation power the sub-tasks generation rhythm may be adapted to maintain a regular workload both for the CPU cores and GPU threads: some CPU cores, not involved in the sub-tasks generation, could be made available for sub-tasks computing. This leads to the 3-level parallelism scheme presented in Fig. 8, where p, q and r respectively correspond to: (p) the server-level tasks generation depth, (q) the client-level sub-tasks generation one, (r) the remaining depth in the tree evaluation, *i.e.* the number of remaining variables to be set before reaching the leaves.

***Backtrack* and *regularized* methods hybridization -** The Backtrack version of the Miller algorithm suits CPU execution and allows to cut branches during the tree evaluation, reducing the search space and limiting the combinatorial explosion effects. A regularized version must be developed, since GPUs execution requires synchronous execution of the threads, with as few branching divergence as possible; however, this method imposes to browse the entire search space and is too time-consuming.
We propose to hybridize two methods in order to take advantage of both of them for the multiGPU parallel execution: for tasks and sub-tasks generated at sever and client levels, the tree development by the CPU cores is made using the backtrack method, cutting branches as soon as possible [and generating only possible tasks]; when computing the sub-tasks generated at client-level, the CPU cores involved in the sub-tasks resolution use the backtrack method and the GPU threads the regularized one.

2.2. Experiments tuning

In order to take advantage of all the computing power of the GPU we have to refine the way we use them: this section presents the experimental study required to choose optimal settings. This tuning allowed us to prove our proposal on significant instances of the Langford problem.

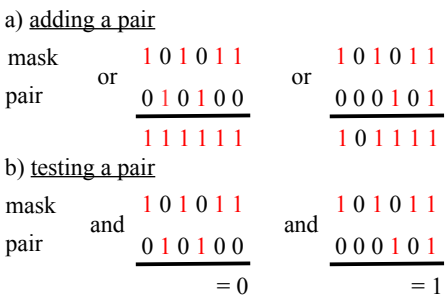


Figure 7. Testing and adding position

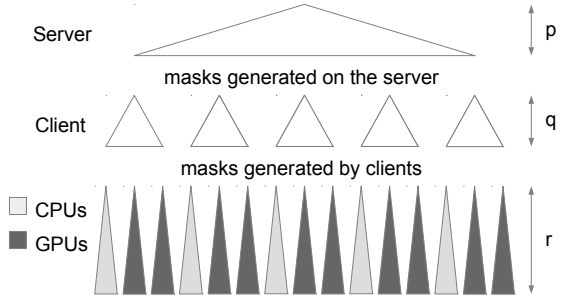


Figure 8. Server client distribution

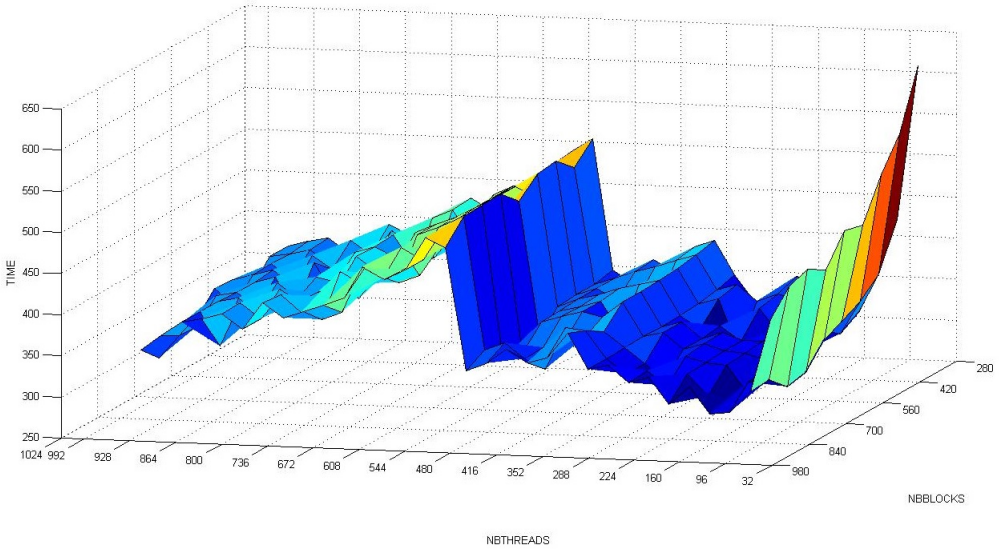


Figure 9. Time depending on grid and block size on $n = 15$

2.2.1. Registers, blocks and grid

In order to use all GPUs capabilities, the first way was to fill the blocks and a grid. To maximize occupancy (ratio between active warps and the total number of warps) NVIDIA suggests to use 1024 threads per block to improve GPU performances and proposes a CUDA occupancy calculator[4]. But, confirmed by the Volkov's results [14], we experimented that better performances may be obtained using lower occupancy. Indeed, another critical criterion is the inner GPU registers occupation. The optimal number of registers (57 registers) is obtained by setting 9 pairs placed on the client for $L(2, 15)$, thus 6 pairs are remaining for GPU computation.

In order to tune the blocks and grid sizes, we performed tests on the ROMEO architecture. Fig. 9 represents the time in relation with a number of blocks per grid and a number of threads per block. The most relevant result, observed as a local minimum on the 3D surface, is obtained near 64 or 96 threads per block; for the grid size, the limitation is relative to the GPU global memory size. It can be noted that we do not need shared memory because their are no data exchanges between threads. This allows us to use the total available memory for the L1 cache for each thread.

[4] `http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls`

2.2.2. Streams

A client has to prepare work for GPU. There are four main steps: generate tasks, load them into the device memory, process the task on the GPU and then get results.

CPU-GPU memory transfers cause huge time penalties (about 400 cycles latency for transfers between CPU memory and GPU *device memory*). At first, we had no overlapping between memory transfer and kernel computation because the tasks generation on CPU was too long compared to the kernel computation. To reduce the tasks generation time we used OpenMP in order to use eight available CPU cores. Thus, CPU computation was totally hidden by memory transfers and GPU kernel computation. We tried using up to 7 streams; as shown by Fig. 10, using only two simultaneous streams did not improve efficiency, because the four steps did not overlap completely; the best performances were obtained with three streams; the slow increase in the next values is caused by synchronization overhead and CUDA streams management.

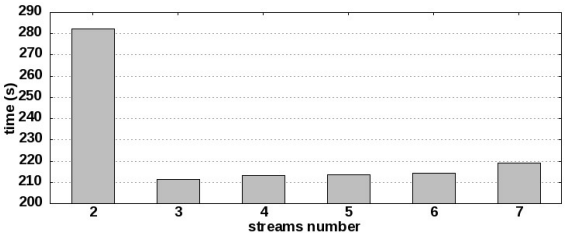


Figure 10. Computing time depending on streams number

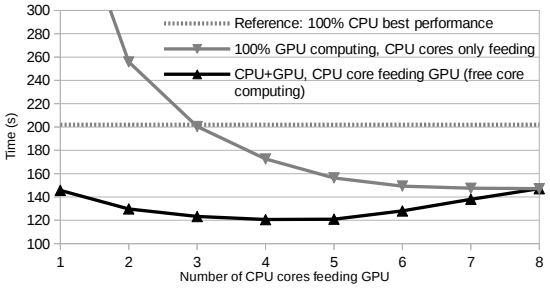


Figure 11. CPU cores optimal distribution for GPU feeding

2.2.3. Setting up the server, client and GPU depths

We now have to set the depths of each actor, server (p), client (q) and GPU (r) (see Fig. 8).

First we set the $r = 5$ for large instances because of the GPU limitation in terms of registers by threads, exacerbated by the use of numerous $64bits$ integers. For $r \geq 6$, we get too many registers (64) and for $r \leq 4$ the GPU computation is too fast compared to the memory load overhead.

Clients are the buffers between the server and the GPUs: $q = n - p - r$. So we have conducted tests by varying the server depth, p. The best result is obtained for $p = 3$ and performance decreases quickly for higher values. This can be explained since more levels on the server generates smaller tasks; thus GPU use is not long enough to overlap memory exchanges.

2.2.4. CPU: Feed the GPUs and compute

The first work of CPU cores is to prepare tasks for GPU so that we can generate overlapping between memory load and kernel computation. In this configuration using eight cores to generate GPU tasks under-uses CPU computation power. It is the reason why we propose to use some of the CPU cores to take part of the sub-problems treatment. Fig. 11 represents computation time in relation with different task distributions between CPU and GPU. We experimentally demonstrated that only 4 or 5 CPU cores are enough to feed GPU, the other ones can be used to perform backtrack resolution in competition with GPUs.

2.3. Results

2.3.1. Regularized method results

We now can show the results obtained for our massively parallel scheme using the previous optimizations, comparing the computation times of successive instances of the Langford problem. These tests were performed on 20 nodes of the ROMEO supercomputer, hence 40 CPU/GPU machines.

The previous limit with Miller's algorithm was $L(2, 19)$, obtained in 1999 after 2.5 years of sequential effort and at the same time after 2 months with a distributed approach [3]. Our computation scheme allowed us to obtain it in less than 4 hours (Table 2), this being not only due to Moore law progress.
Note that the computation is 1.6 faster with CPU+GPU together than using 8 CPU cores. In addition, the GPUs compute 200000× more nodes of the search tree than the CPUs, with a faster time.

Table 2. Regularized method (seconds)

n	CPU (8c)	GPU (4c) + CPU (4c)
15	2.5	1.5
16	21.2	14.3
17	200.3	120.5
18	1971.0	1178.2
19	22594.2	13960.8

Table 3. Backtrack (seconds)

n	CPU (8c)	GPU (4c) + CPU (4c)
17	29.8	7.3
18	290.0	73.6
19	3197.5	803.5
20	–	9436.9
21	–	118512.4

The computation time between two different consecutive instances being multiplied by 10 approximately, this could allow us to obtain $L(2, 20)$ in a reasonable time.

2.3.2. Backtracking on GPUs

It appears at first sight that using backtracking on GPUs without any regularization is a bad idea due to threads synchronization issues. But in order to compare CPU and GPU computation power in the same conditions we decide to implement the original backtrack method on GPU (see Fig. 3) with only minor modifications. In these conditions we observe very efficient work of the NVIDIA scheduler, which perfectly handles threads desynchronization. Thus we use the same server-client distribution as in 2.1.4, each client generates masks for both CPU and GPU cores. The workload is then statically distributed on GPU and CPU cores. Executing the backtrack algorithm on a randomly chosen set of sub-problems allowed us to set the GPU/CPU distribution ratio experimentally to 80/20%,

The experiments were performed on 129 nodes of the ROMEO supercomputer, hence 258 CPU/GPU machines and one node for the server. Table 3 shows the results with this configuration. This method first allowed us to perform the computation of $L(2, 19)$ in less than 15 minutes, 15× faster than with the regularized method; then, we pushed the limitations of the Miller algorithm up to $L(2, 20)$ in less than 3 hours and even $L(2, 21)$ in about 33 hours[5].

[5]Even if this instance has no interest since it is known to have no solution

This exhibits the ability of the GPU scheduler to manage highly irregular tasks. It proves that GPUs are adapted even to solve combinatorial problems, which they were not supposed to do.

3. Godfrey's algebraic method

The previous part presents the Miller algorithm for the Langford problem, this method cannot achieve bigger instances than the $L(2, 21)$.
An algebraic representation of the Langford problem has been proposed by M. Godfrey in 2002. In order to break the limitation of $L(2, 24)$ we already used this very efficient problem specific method. In this part we describe this algorithm and optimizations, and our implementation on multiGPU clusters.

3.1. Method description

Consider $L(2, 3)$ and $X = (X_1, X_2, X_3, X_4, X_5, X_6)$. It proposes to modelize $L(2, 3)$ by $F(X, 3) = (X_1X_3 + X_2X_4 + X_3X_5 + X_4X_6) \times (X_1X_4 + X_2X_5 + X_3X_6) \times (X_1X_5 + X_2X_6)$

In this approach each term represents a position of both cubes of a given color and a solution to the problem corresponds to a term developed as $(X_1X_2X_3X_4X_5X_6)$; thus the number of solutions is equal to the coefficient of this monomial in the development. More generally, the solutions to $L(2, n)$ can be deduced from $(X_1X_2X_3X_4X_5...X_{2n})$ terms in the development of $F(X, n)$.

If $G(X, n) = X_1...X_{2n}F(X, n)$ then it has been shown that:

$$\sum_{(x_1,...,x_{2n})\in\{-1,1\}^{2n}} G(X, n)_{(x_1,...,x_{2n})} = 2^{2n+1}L(2, n)$$

So

$$\sum_{(x_1,...,x_{2n})\in\{-1,1\}^{2n}} \Big(\prod_{i=1}^{2n} x_i\Big) \prod_{i=1}^{n} \sum_{k=1}^{2n-i-1} x_k x_{k+i+1} = 2^{2n+1}L(2, n)$$

That allows to get $L(2, n)$ from polynomial evaluations. The computational complexity of $L(2, n)$ is of $O(4^n \times n^2)$ and an efficient big integer arithmetic is necessary. This principle can be optimized by taking into account the symmetries of the problem and using the Gray code: these optimizations are described below.

3.2. Optimizations

Some works focused on finding optimizations for this arithmetic method [15]. Here we explain the symmetric and computation optimizations used in our algorithm.

3.2.1. Evaluation parity

As $[F(-X, n) = F(X, n)]$, G is not affected by a global sign change. In the same way the global sign does not change if we change the sign of each pair or impair variable.

Using these optimizations we can set the value of two variables and accordingly divide the computation time and result size by four.

3.2.2. Symmetry summing

In this problem we have to count each solution up to a symmetry; thus for the first pair of cubes we can stop the computation at half of the available positions considering $S'_1(x) = \sum_{k=1}^{n-1} x_k x_{k+2}$ instead of $S_1(x) = \sum_{k=1}^{2n-2} x_k x_{k+2}$. The result is divided by 2.

3.2.3. Sums order

Each evaluation of $S_i(x) = \sum_{k=1}^{2n-i-1} x_k x_{k+i+1}$, before multiplying might be very important regarding to the computation time for this sum. Changing only one value of x_i at a time, we can recompute the sum using the previous one without global recomputation. Indeed, we order the evaluations of the outer sum using Gray code sequence. Then the computation time is considerably reduced.

Based on all these improvements and optimizations we can use the Godfrey method in order to solve huge instances of the Langford problem. The next section develops the main issues of our multiGPU architecture implementation.

3.3. Implementation details

In this part we present specific adaptations required to implement the Godfrey method on a multiGPU architecture.

3.3.1. Optimized big integer arithmetic

In each step of computation, the value of each S_i can reach $2n-i-1$ in absolute value, and their product can reach $\frac{(2n-2)!}{(n-2)!}$. As we have to sum the S_i product on 2^{2n} values, in the worst case we have to store a value up to $2^{2n}\frac{(2n-2)!}{(n-2)!}$, which corresponds to 10^{61} for $n = 28$, with about 200 bits.

So we need few big integer arithmetic functions. After testing existing libraries like GMP for CPU or CUMP for GPU, we have come to the conclusion that they implement a huge number of functionalities and are not really optimized for our specific problem implementation: product of "small" values and sum of "huge" values.

Finally, we developed a light CPU and GPU library adapted to our needs. In the sum for example, as maintaining carries has an important time penalty, we have chosen to delay the spread of carries by using buffers: carries are accumulated and spread only when useful (for example when the buffer is full). Fig. 12 represents this big integer handling.

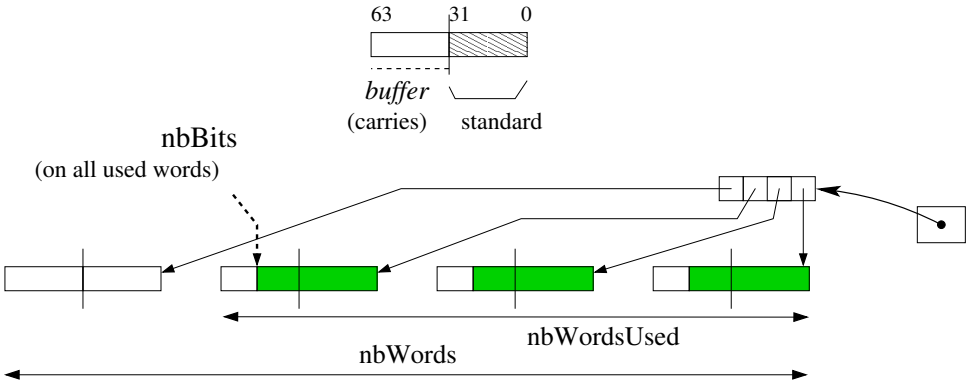


Figure 12. Big integer representation, 64 bits words

3.3.2. Gray sequence in memory

The Gray sequence cannot be stored in an array, because it would be too large (it would contain 2^{2n} byte values). This is the reason why only one part of the Gray code sequence is stored in memory and the missing terms are directly computed from the known ones using arithmetic considerations. The size of the stored part of the Gray code sequence is chosen to be as large as possible to be contained in the processor's cache memory, the L1 cache for the GPUs threads: so the accesses are fastened and the computation of the Gray code is optimized. For an efficient use of the E5-2650 v2 ROMEO's CPUs, which disposes of 20 MB of level-3 cache, the CPU Gray code sequence is developed recursively up to depth 25. For the K20Xm ROMEO's GPUs, which dispose of 8 KB of constant memory, the sequence is developed up to depth 15. The rest of the memory is used for the computation itself.

3.3.3. Tasks generation and computation

In order to perform the computation of the polynomial, two variables can be set among the $2n$ available. For the tasks generation we choose a number p of variables to generate 2^p tasks by developing the evaluation tree to depth p.

The tasks are spread over the cluster, either synchronously or asynchronously.

Synchronous computation - The first experiment was carried out with an MPI distribution of tasks of the previous model. Each MPI process finds its tasks list based on its process id; then converting each task number into binary gives the task's initialization. The processes work independently; finally the root process ($id = 0$) gathers all the computed numbers of solutions and sums them.

Asynchronous computation - In this case the tasks can be computed independently. As with the synchronous computation, the tasks' initializations are retrieved from their number. Each machine can get a task, compute it, and then store its result; then when all the tasks have been computed, the partial sums are added together and the total result is provided.

3.4. Experimental settings

This part presents the experimental context and methodology, and the way experiments were carried out. This study has similar goals as for the Miller's resolution experiments.

3.4.1. Experimental methodology

We present here the way the experimental settings were chosen. Firstly, we define the tasks distribution, secondly, we set the number of threads per GPU block; finally, we set the CPU/GPU distribution.

Tasks distribution depth - This value being set it is important to get a high number of blocks to maintain sufficient GPU load. Thus, we have to determine the best number of tasks for the distribution. As presented in part 3.3.3, the number p of bits determines 2^p tasks. On the one hand, too many tasks are a limitation for the GPU that cannot store all the tasks in its 6GB memory. On the other hand, not enough tasks mean longer tasks and too few blocks to fill the GPU grid. Fig. 14 shows that for the $L(2, 23)$ instance the best task number is with generation depth 28.

Number of threads per block - In order to take advantage of the GPU computation power, we have to determine the threads/block distribution. Inspired by our experiments with Miller's algorithm we know that the best value may appear at lower occupancy. We perform tests on a given tasks set varying the threads/block number and grid size associated. Fig. 13 presents the tests performed on the $n = 20$ problem: the best distribution is around 128 threads per block.

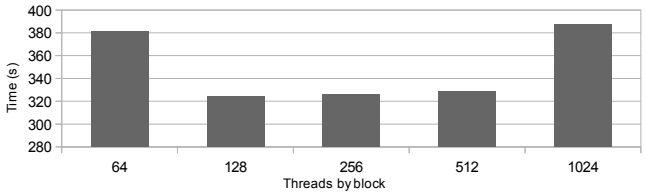


Figure 13. $L(2, 20)$, number of threads per block

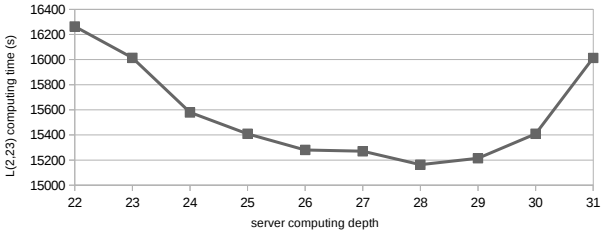


Figure 14. Influence on server generation depth

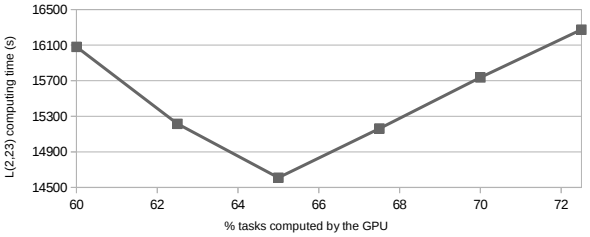


Figure 15. Influence of tasks repartition

CPU vs GPU distribution - The GPU and CPU computation algorithm will approximately be the same. In order to take advantage of all the computational power of both components we have to balance tasks between CPU and GPU. We performed tests by changing the CPU/GPU distribution based on simulations on a chosen set of tasks. Fig. 15 shows that the best distribution is obtained when the GPU handles 65% of the tasks. This optimal load repartition directly results from the intrinsics computational power of each component; this repartition should be adapted if using a more powerful GPU like Tesla K40 or K80.

3.4.2. Computing context

As presented in part 1.2, we used the ROMEO supercomputer to perform our tests and computations. On this supercomputer SLURM [16] is used as a reservation and a job queue manager. This software allows two reservation modes: a static one-job limited reservation or an opportunity to dynamically submit several jobs in a Best-Effort manner.

Static distribution - In this case we used the synchronous distribution presented in 3.3.3. We submited a reservation with the number of MPI processes and the number of cores per process. This method is useful to get the results quickly if we can get at once a large amount of computation resources. It was used to perform the computation of small problems, and even for $L(2, 23)$ and $L(2, 24)$.

As an issue, it has to be noted that it is difficult to quickly obtain a very large reservation on such a shared cluster, since many projects are currently running.

Best effort - SLURM allows to submit tasks in the specific Best-Effort queue, which does not count in the user *fair-share*. In this queue, if a node is free and nobody is using it, the

reservation is set for a job in the best effort queue for a minimum time reservation. If another user asks for a reservation and requests this node, the best effort job is killed (with, for example, a SIGTERM signal). This method, based on asynchronous computation, enables a maximal use of the computational resources without blocking for a long time the entire cluster.

For $L(2,27)$ and even more for $L(2,28)$ the total time required is too important to use the whole machine off a challenge period, thus we chose to compute in a Best-Effort manner. In order to fit with this submission method we chose a reasonable time-per-task, sufficient to optimize the treatments with low loading overhead, but not too long so that killed tasks are not too penalizing for the global computation time. We empirically chose to run 15-20 minute tasks and thus we considered $p = 15$ for $n = 27$ and $p = 17$ for $n = 28$.

The best effort based algorithm is presented on Fig. 16. The task handler maintains a maximum of 256 tasks in the queue; in addition the entire process is designed to be fault-tolerant since killed tasks have to be launched again. When finished, the tasks generate an ouput containing the number of solutions and computation time, that is stored as a file or database entry. At the end the outputs of the different tasks are merged and the global result can be provided.

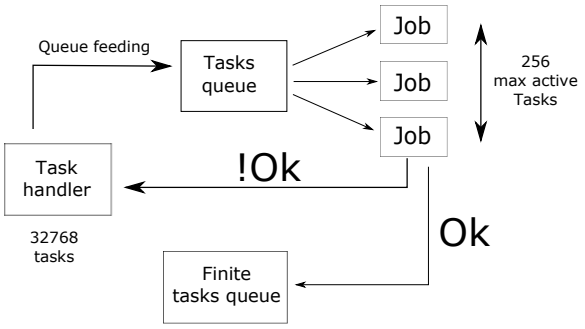


Figure 16. Best-effort distribution

3.5. Results

After these optimizations and implementation tuning steps, we conducted tests on the ROMEO supercomputer using best-effort queue to solve $L(2,27)$ and $L(2,28)$. We started the experiment after an update of the supercomputer, that implied a cluster shutdown. Then the machine was restarted and was about 50% idle for the duration of our challenge. The computation lasted less than 2 days for $L(2,27)$ and 23 days for $L(2,28)$. The following describes performances considerations.

Computing effort - For $L(2,27)$, the effective computation time of the 32,768 tasks was about 30 million seconds (345.4 days), and 165,000" elapsed time (1.9 days); the average time of the tasks was 911", with a standard deviation of 20%. For the $L(2,28)$ 131,072 tasks the total computation time was about 1365 days (117 million seconds), as 23 day elapsed time; the tasks lasted 1321" on average with a 12% standard deviation.

Best-effort overhead - With $L(2,27)$ we used a specific database to maintain information concerning the tasks: 617 tasks were aborted [by regular user jobs] before finishing (1.9%), with an average computing time of 766" (43% of the maximum requested time for a task). This consumed 472873", which overhead represents 1.6% of the effective computing effort.

Cluster occupancy - Fig. 17 presents the tasks resolution over the two computation days for $L(2, 27)$. The experiment elapse time was 164700" (1.9 days). Compared to the effective computation time, we used an average of 181.2 machines (CPU-GPU couples): this represents 69.7% of the entire cluster.

Fig. 18 presents the tasks resolution flow during the 23 days computation for $L(2, 28)$. We used about 99 machines, which represents 38% of 230 available nodes.

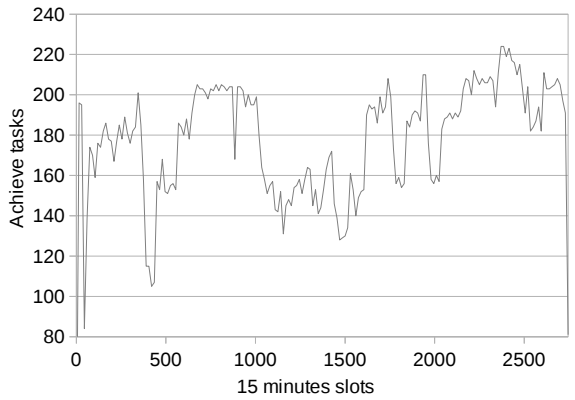


Figure 17. $L(2, 27)$ tasks grouped by 15" slots

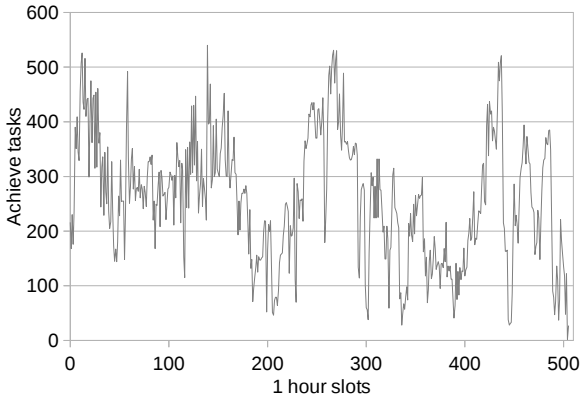


Figure 18. $L(2, 28)$ tasks grouped by 1 hour slots

For $L(2, 27)$, these results confirm that the computation took great advantage of the low occupancy of the cluster during the experiment. This allowed us to obtain a weak best-effort overhead, and an important cluster occupancy. Unfortunately, for $L(2, 28)$ on such a long period we got a lower part of the supercomputer dedicated to our computational project. Thus, we are confident in good perspectives for the $L(2, 31)$ instance if computed on an even larger cluster or several distributed clusters.

Conclusion

This paper presents two methods to solve the Langford pairing problem on multiGPU clusters. In its first part the Miller's algorithm is presented. Then to break the problem limitations we show optimizations and implementation of Godfrey's algorithm.

CSP resolution method - As any combinatorial problem can be represented as a CSP, the Miller algorithm can be seen as a general resolution scheme based on the backtrack tree browsing. A three-level tasks generation allows to fit the multiGPU architecture. MPI or Best-Effort are used to spread tasks over the cluster, OpenMP for the CPU cores distribution and then CUDA to take advantage of the GPU computation power. We were able to compute $L(2, 20)$ with this regularized method and to get an even better time with the basic backtrack. This proves the proposed approach and also exhibits that the GPU scheduler is very efficient at managing highly divergent threads.

MultiGPU clusters and best-effort - In addition and with the aim to beat the Langford limit we present a new implementation of the Godfrey method using GPUs as accelerators. In order to use the supercomputer ROMEO, which is shared by a large scientific community, we have implemented a distribution that does not affect the machine load, using a best-effort queue. The computation is fault-tolerant and totally asynchronous.

Langford problem results - This study enabled us to compute $L(2, 27)$ and $(L2, 28)$ in respectively less than 2 days and 23 days on the University of Reims ROMEO supercomputer. The total number of solutions is:

L(2,27) = 111,683,611,098,764,903,232
L(2,28) = 1,607,383,260,609,382,393,152

Perspectives - This study shows the benefit of using GPUs as accelerators for combinatorial problems. In Miller's algorithm they handle 80% of the computation effort and 65% in Godfrey's. As a near-term prospect, we want to scale and show that it is possible to use the order of 1000 or more GPUs for pure combinatorial problems.
The next step of this work is to generalize the method to optimization problems. This adds an order of complexity since shared information has to be maintained over a multiGPU cluster.

This work was supported by the High Performance Computing Center of the University of Reims Champagne-Ardenne, ROMEO.

References

1. Gardner M. Mathematics, magic and mystery. Dover publication; 1956.
2. Simpson JE. Langford Sequences: perfect and hooked. Discrete Math. 1983;44(1):97–104.
3. Miller JE. Langford's Problem: http://dialectrix.com/langford.html; 1999. Available from: `http://www.lclark.edu/~miller/langford.html`.
4. Walsh T. Permutation Problems and Channelling Constraints. APES Research Group; 2001. APES-26-2001. Available from: `http://www.dcs.st-and.ac.uk/~apes/reports/apes-26-2001.ps.gz`.
5. Smith B. Modelling a Permutation Problem. In: Proceedings of ECAI'2000, Workshop on Modelling and Solving Problems with Constraints, RR 2000.18. Berlin; 2000. Available from: `http://www.dcs.st-and.ac.uk/~apes/2000.html`.
6. Larsen J. Counting the number of Skolem sequences using inclusion exclusion. 2009;.
7. Assarpour A, Barnoy A, Liu O. Counting the Number of Langford Skolem Pairings; 2015. .
8. Nvidia C. Compute unified device architecture programming guide. 2007;.
9. Kirk DB, Wen-mei WH. Programming massively parallel processors: a hands-on approach. Newnes; 2012.
10. Habbas Z, Krajecki M, Singer D. Parallelizing Combinatorial Search in Shared Memory. In: Proceedings of the fourth European Workshop on OpenMP. Roma, Italy; 2002. .
11. Montanari U. Networks of Constraints: Fundamental Properties and Applications to Pictures Processing. Information Sciences. 1974;7:95–132.
12. Prosser P. Hybrid algorithms for the constraint satisfaction problem. Computational intelligence. 1993;9(3):268–299.
13. Krajecki M. An object oriented environment to manage the parallelism of the FIIT applications. In: Parallel Computing Technologies. Springer; 1999. p. 229–235.

14. Volkov V. Better performance at lower occupancy. In: Proceedings of the GPU Technology Conference, GTC. vol. 10. San Jose, CA; 2010. p. 16.

15. Jaillet C. In french: Résolution parallèle des problèmes combinatoires [PhD]. Université de Reims Champagne-Ardenne, France; 2005.

16. Jette M, Grondona M. SLURM : Simple Linux Utility for Resource Management; June 23, 2003.

DOI: 10.14529/jsfi160203

A Radical Approach to Computation with Real Numbers

John L. Gustafson[1]

If we are willing to give up compatibility with IEEE 754 floats and design a number format with goals appropriate for 2016, we can achieve several goals simultaneously: Extremely high energy efficiency and information-per-bit, no penalty for decimal operations instead of binary, rigorous bounds on answers without the overly pessimistic bounds produced by interval methods, and unprecedented high speed up to some precision. This approach extends the ideas of unum arithmetic introduced two years ago by breaking completely from the IEEE float-type format, resulting in fixed bit size values, fixed execution time, no exception values or "gradual underflow" issues, no wasted bit patterns, and no redundant representations (like "negative zero"). As an example of the power of this format, a difficult 12-dimensional nonlinear robotic kinematics problem that has defied solvers to date is quickly solvable with absolute bounds. Also unlike interval methods, it becomes possible to operate on arbitrary disconnected subsets of the real number line with the same speed as operating on a simple bound.

Keywords: Floating point, Unum computing, Computer arithmetic, Energy efficiency, Valid arithmetic.

1. A Quick Overview of "Type 1 unums"

The ***unum*** (***u***niversal ***num***ber) arithmetic system was presented publicly in 2013, and a text describing the approach was published in 2015 [2]. As originally defined, a unum is a superset of IEEE 754 floating-point format [6] that tracks whether a number is an exact float or lies in the open interval one Unit of Least Precision (ULP) wide, between two exact floats. While the meaning of the sign, exponent, and fraction bit fields take their definition from the IEEE 754 standard, the bit lengths of the exponent and fraction are allowed to vary, from as small as a single bit up to some maximum that is set in the environment. The formats are contrasted in Fig. 1.

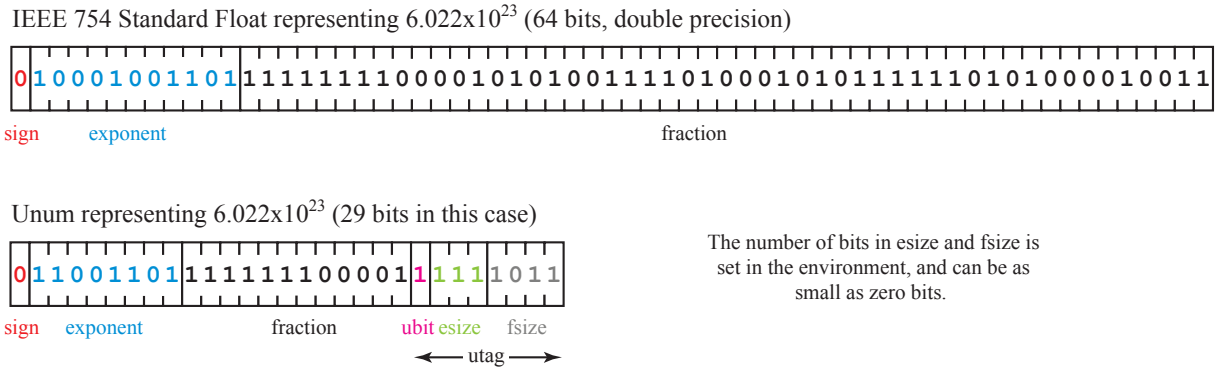


Figure 1. Comparison of IEEE 754 64-bit float and a Type 1 unum representing the same value

The inclusion of the "uncertainty bit" (ubit) at the end of the fraction eliminates rounding, overflow, and underflow by instead tracking when a result lands between representable floats. Instead of underflow, the ubit marks it as lying in the open interval between zero and the smallest nonzero number. Instead of overflow, the ubit marks it as lying in the open interval between the maximum finite float value and infinity.

[1]A*STAR Computational Resources Center and National University of Singapore (joint appointment)

Flexible dynamic range and precision eliminates the pressure for a programmer to choose a "one size fits all" data type. Programmers typically choose double precision (64-bit) as over-insurance against numerical problems; however, this typically wastes storage and bandwidth by about a factor of two or more since the maximum precision is only needed for some fraction of the calculations. Furthermore, high precision is no guarantee against catastrophic errors [5]. As Fig. 1 shows, adding self-descriptive bits (the "utag") to the float format can actually *save* total bit count, much the same way that having an exponent field in a float saves bits compared to an integer or fixed-point representation.

The unum definition in [2] also fixes some problems with the IEEE 754 Standard, such as negative zero, the wasting of trillions of unique bit patterns for representing "Not-a-Number" (NaN), and the failure to guarantee bit-identical results from different computer systems. However, the original "Type 1 unums" have drawbacks, particularly for hardware implementation:

- They either use variable storage size or must be unpacked into a fixed storage size that includes some unused bits, much like managing variable file sizes in mass storage.
- The utag adds a level of indirection; it must be read first to reference the other fields.
- The logic involves more conditional branches than floats.
- Some values can be expressed in more than one way, and some bit patterns are not used.

2. The Ideal Format

The current IEEE 754 format evolved from historical formats and years of committee discussions selecting compromises between speed and correctness. What if we abandon ties to the past and ask what would be an *ideal* representation of real number values? Here are some goals, some of which are similar to mathematical goals defined in [3]

- All arithmetic operations would be equally fast.
- There would be no performance penalty for using decimal representation.
- It would be easy to build using current processor technology.
- There would be no exceptions like subnormal numbers, NaN, or "negative zero."
- Accuracy could be managed automatically by the computer.
- No real numbers would be overlooked; there might be limited accuracy, but no omissions.
- The system would be mathematically sound, with no rounding errors.

The last three goals are achieved by Type 1 unums, so we want to preserve those advantages. A value like π can be represented honestly as $3.14\cdots$ where the "$\cdots$" indicates that the value lies in the open interval between two exact numbers, (3.14, 3.15). The error of non-unum number systems is in restricting representations to exact values *sampled* from the real number line; when an algorithm exposes the mathematical omission, the strategy to date has been to demand more exact points to fill in the gaps, a futile chase that ignores inherent properties of the real number line. Once a system embraces the idea that finite-state representations can represent exact values **and** the open ranges between exact values, there is hope of creating a bijection: that is, a mapping from bit strings to a partitioning of the extended reals that is one-to-one and onto. Furthermore, it may happen that a surprisingly low-precision bit string has more expressive power than a high-precision format that can represent only a set of exact values.

3. A Mapping to the Projective Real Line, and its Power Set

3.1. The projective reals resemble modern signed (two's complement) integers

One representation of the real line bends the two infinite extremes into a circle. The distinction between positive and negative infinity is lost, replaced by a single "point at infinity." We can label it $\pm\infty$ for now, and think of it as the *reciprocal of zero*. A mapping of the projective reals to two-bit integer strings is shown in Fig. 2.

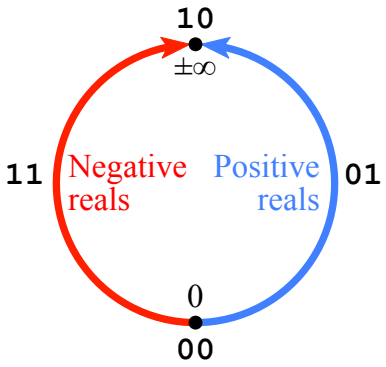


Figure 2. Projective real number line and two-bit two's complement integers

As signed integers, the four strings around the circle **00, 01, 10** and **11** correspond to two's complement integers $0, 1, -2, -1$. They "wrap" from positive to negative at exactly the point where the projective reals wrap from positive to negative. We call these two-bit strings *unum* values because they represent either exact quantities or the open interval between exact quantities. For our purposes, we choose to treat $\pm\infty$ as if it were an exact quantity.

3.2. SORNs and the elimination of "indeterminate forms"

Imagine another bit string now, one that expresses the absence (**0**) or presence (**1**) of each of the sets shown in Fig. 2. That is, the power set of the four subsets of the projective real numbers, $\pm\infty$, $(-\infty, 0)$, 0 and $(0, \infty)$. Such a bit string allows us to operate on subsets of the reals using bit-level arithmetic, as opposed to symbol manipulation. For example, the bit vector **0011** means the absence of $\pm\infty$ and $(-\infty, 0)$ and the presence of 0 and $(0, \infty)$ in the set, which we could write in more conventional notation as the interval $[0, \infty)$. Call this bit string a *SORN*, for Sets Of Real Numbers. To make them easier to distinguish from the binary numbers that use positional notation to represent integers, we can use the following shapes: ● ○ ▮ ▯. A circle represents an open interval and a thin rectangle represents an exact value. They are filled if present and hollow if absent. If we have access to color display, then it further helps read SORNs if negative values are red and positive values are blue; the ▮ or ▯ shapes for values 0 and $\pm\infty$ remain black since they have no sign. Here is a table of the 16 possible SORN values for the unum set shown in Fig. 2:

Table 1. Four-bit SORN values

SORN notation	English description	Math notation
▯○▯○	The empty set	$\{\}$ or $\varnothing$
▯○▯●	All positive reals	$(0, \infty)$
▯○▮○	Zero	0
▯○▮●	All nonnegative reals	$[0, \infty)$
▯●▯○	All negative reals	$(-\infty, 0)$
▯●▯●	All nonzero reals	$(-\infty, 0) \cup (0, \infty)$
▯●▮○	All nonpositive reals	$(-\infty, 0]$
▯●▮●	All reals	$(-\infty, \infty)$ or $\mathbb{R}$
▮○▯○	The point at infinity	$\pm\infty$
▮○▯●	The extended positive reals	$(0, \infty]$
▮○▮○	The unsigned values	$0 \cup \pm\infty$
▮○▮●	The extended nonnegative reals	$[0, \infty]$
▮●▯○	The extended negative reals	$[-\infty, 0)$
▮●▯●	All nonzero extended reals	$[-\infty, 0) \cup (0, \infty]$
▮●▮○	The extended nonpositive reals	$[-\infty, 0]$
▮●▮●	All extended reals	$[-\infty, \infty]$ or $\mathbb{R}^+$

Because we have the reals on a circle instead of a line, it is possible to notate the nonzero extended reals, for example, as the interval "$(0, 0)$" instead of having to write $[-\infty, 0) \cup (0, \infty]$. Usually, an interval is written as a closed interval $[x, y]$ where $x \leq y$, or one with open endpoints $(x, y]$, $[x, y)$, (x, y) where $x < y$. Any contiguous set can be written as a simple interval, where numbers increase from the left endpoint until they reach the right endpoint, even if it means passing through $\pm\infty$. This is similar to the idea of "outer intervals" [4]. Also, for the reader who is already missing the values $-\infty$ and $+\infty$ that IEEE floats provide, notice that **we still have them**, in the form of unbounded intervals flanking $\pm\infty$. For example, we could take the logarithm of $(0, \infty)$ and obtain $(-\infty, \infty)$. The square of that result would be $[0, \infty)$, and so on. If a SORN shows the presence of $\pm\infty$ and one of the two unbounded unums $(1, \infty)$ or $(-\infty, -1)$, we treat it as a closed endpoint, "$\infty]$" on the right, or "$[-\infty$" on the left.

The arithmetic tables for $+ \; - \; \times \; \div$ on these SORN values look at first like a hellish collection of **all the things you are never supposed to do**: zero divided by zero, infinity minus infinity, and other so-called *indeterminate forms*. They are called indeterminate because they do not produce *single numbers*. There is usually some wringing of hands about dividing nonzero numbers by zero as well; is the answer $+\infty$ or $-\infty$? We have no such difficulties here. If we take the limit of, say, $x - y$ as $x \to \infty$ and $y \to \infty$, we find it can be any value, and there is a SORN for that: ▮●▮●. Similarly, divide any positive number by x as $x \to 0$ and the result is ∞ or $-\infty$ depending on whether the limit is from the left or the right. However, we have just the thing for that situation: $\pm\infty$, represented by ▮○▯○. There is no reason to wish for a NaN representation.

Looking ahead a bit, we can imagine taking the square root of the negative reals, ▯●▯○. With conventional floats we would certainly have to throw up our hands and return a NaN. With SORNs, the answer is the empty set, ▯○▯○. We can even take care of indeterminate forms like 1^∞, using limits of x^y as $x \to 1$ (from above or below) and $x \to \infty$. It is simply the nonnegative extended reals, ▮○▮●. There is nothing wrong with the result of a calculation being

a set, including the empty set. We do not need to admit defeat by declaring something NaN, and in fact can continue calculating. Even if it appears that all information has been lost and the answer can be anything, that is, ▮●▮●, if the next operation were to square the SORN it would result in the nonnegative extended reals ▮○▮●, which has some information about the answer.

3.3. Fast calculation of SORNs with bitwise OR operations

Imagine that we have filled out the addition table for the four unum values $\pm\infty$, $(-\infty, 0)$, 0 and $(0, \infty)$. We can express each unum as a SORN with just one of the four shapes filled in. As tab. 2 shows, addition sometimes produces a SORN with more than one shape filled in.

Table 2. The addition table for two-bit unum inputs and SORN outputs

+	▮○▯○	▯●▯○	▯○▮○	▯○▯●
▮○▯○	▮●▮●	▮○▯○	▮○▯○	▮○▯○
▯●▯○	▮○▯○	▯●▯○	▯●▯○	▯●▮●
▯○▮○	▮○▯○	▯●▯○	▯○▮○	▯○▯●
▯○▯●	▮○▯○	▯●▮●	▯○▯●	▯○▯●

The highlighted parts of the table indicate *information loss*; three of the entries "blur" in that they do not produce a single unum output from two unum inputs. In some systems, this would entail recording a variable amount of data for table entries. With SORNs, all entries are the same number of bits, facilitating table look-up in a computer.

Furthermore, they lend themselves to very fast and simple evaluation of SORN operations with logic OR gates and a bit of parallelism, as shown in Fig. 3.

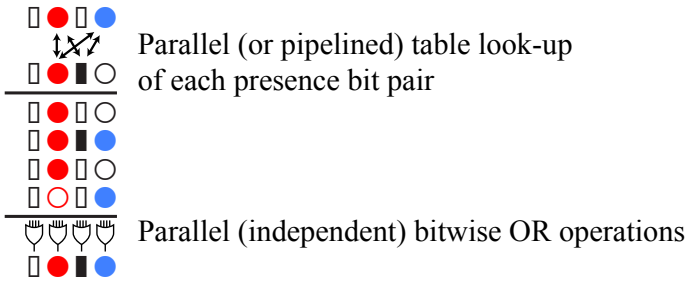


Figure 3. Fast SORN operation using parallel table look-up followed by parallel bitwise OR

The logic gate delay time at current 14 nm technology is about 10 picoseconds. A table lookup involves three gate delays and the parallel OR operation adds another 10 picoseconds, depending on the fan-in design rules. This suggests that scalar operations on the real number line at this ultra-low accuracy level can be done at around 25 GHz. Fig. 4 shows how simple a ROM is for table look-up; the black dots are wired connections requiring no transistors. If the table were stored in DRAM, it would require 3.2 times as many transistors (not counting refresh circuitry) with one transistor per bit. SRAM requires six transistors per bit, which would take 14 times as many transistors.

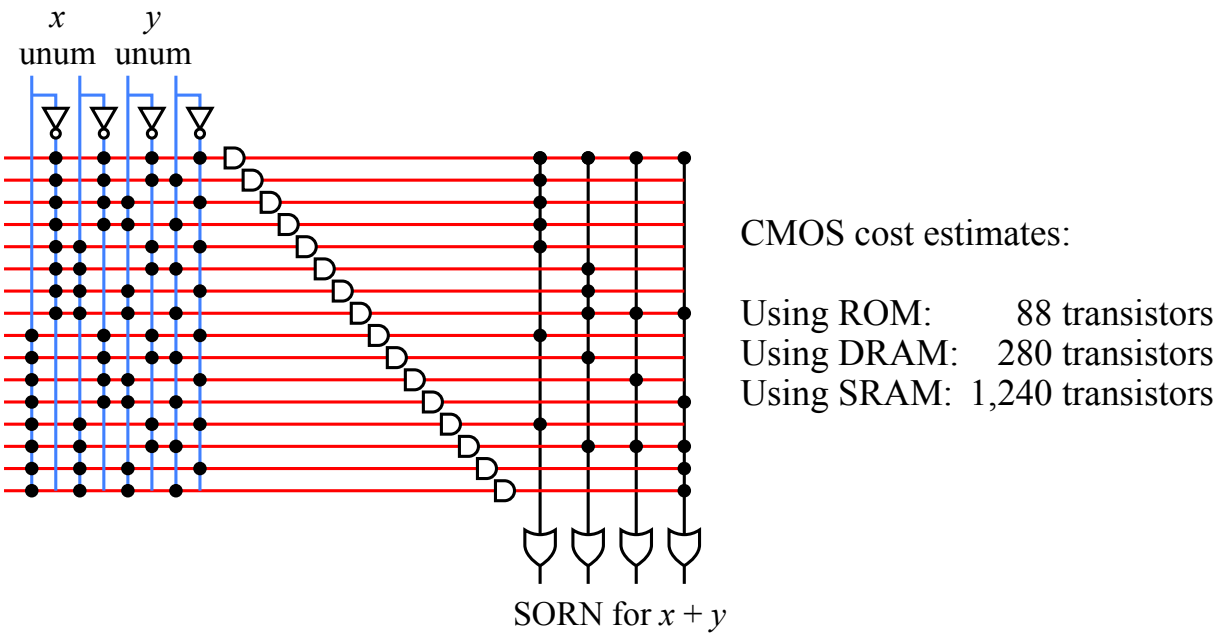


Figure 4. ROM circuit example for the table look-up of the preceding SORN addition

The next step is to start ramping up the accuracy of the unum lattice and the SORNs that go with that lattice.

3.4. The start of a useful number system: a kinematics application

Append another bit to the unum so that we can represent +1 and −1, and the open intervals surrounding those exact numbers. The annotated circle of real values and some examples of SORN representations are shown in Fig. 5. When assigned to binary strings, the first bit resembles a sign bit like that of IEEE floats, though we ignore it for values 0 and ±∞. The *last* bit serves as the "uncertainty bit" or *ubit*, exactly as it did with the IEEE-compatible Type 1 unums definition. The ubit is **0** for exact unums, **1** for open ranges between exact numbers ("inexact" unums, for short). Hence, we color-code those two bits the same way as the original unums [2], with the sign bit in red and the ubit in magenta.

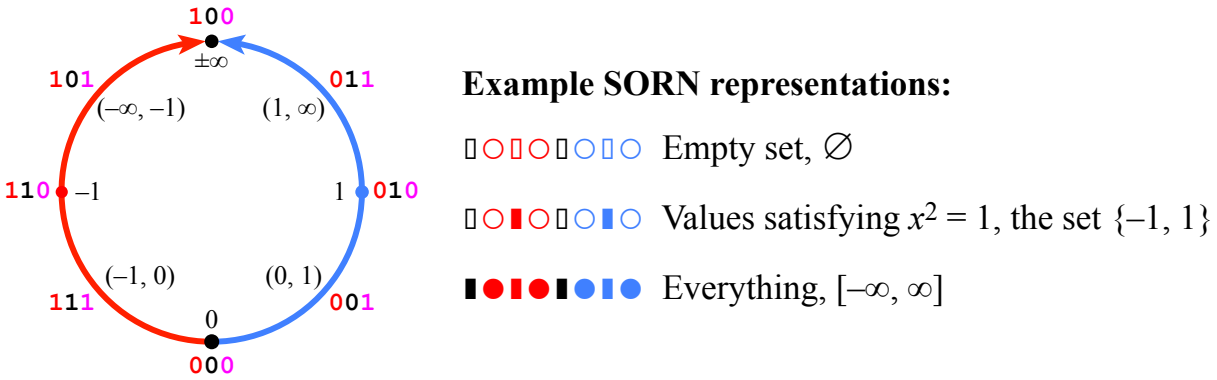
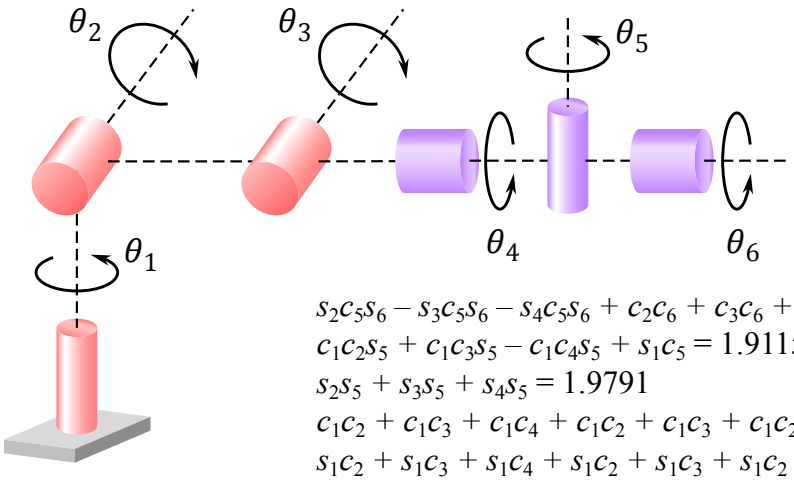


Figure 5. Three-bit unum representation with −1 and +1, and examples of SORNs

Such low precision can be surprisingly useful, since it is often helpful in the early stages of solving a problem to know at least a little bit about where to look for a solution. Are the solutions of bounded magnitude? Are they known to be positive? For example, a classic problem

in robotics is to solve the inverse kinematics of an elbow manipulator [1]. Such a problem and the twelve nonlinear equations in twelve unknowns that it gives rise to are shown in Fig. 6.



$$s_2c_5s_6 - s_3c_5s_6 - s_4c_5s_6 + c_2c_6 + c_3c_6 + c_4c_6 = 0.4077$$
$$c_1c_2s_5 + c_1c_3s_5 - c_1c_4s_5 + s_1c_5 = 1.9115$$
$$s_2s_5 + s_3s_5 + s_4s_5 = 1.9791$$
$$c_1c_2 + c_1c_3 + c_1c_4 + c_1c_2 + c_1c_3 + c_1c_2 = 4.0616$$
$$s_1c_2 + s_1c_3 + s_1c_4 + s_1c_2 + s_1c_3 + s_1c_2 = 1.7172$$
$$s_2 + s_3 + s_4 + s_2 + s_3 + s_2 = 3.9701$$
$$c_i^2 + s_i^2 = 1 \; (1 \le i \le 6)$$

Figure 6. Inverse kinematics problem: a constrained elbow manipulator

The classic approach to such a set of equations is to guess a starting value for the twelve unknowns and iterate toward a solution, if a solution even exists. There might be multiple solutions, but such an approach will find at most one of them. With our ultra-low precision SORNs in Fig. 5, it becomes feasible to test *the entire 12-dimensional space* for regions where solutions can or cannot exist. Unums of magnitude greater than 1 are ruled out by the $c_i^2 + s_i^2 = 1$ equations; the c and s variables are cosines and sines of the six angles, though we do not need that knowledge for the unum approach to converge quickly. If we split all twelve dimensions into two possible open unums $(-1, 0)$ or $(0, 1)$, there are $2^{12} = 4096$ regions of the space of solutions, which can be examined in parallel in a few nanoseconds using the SORN set shown in Fig. 5. The result is the exclusion of 4000 of the spaces as *infeasible* solution regions, leaving only 96 possibilities for further examination. While this sort of approach has been used with interval arithmetic in the past, those computing environments involve 128 bits per variable (two double-precision endpoints), and very slow, energy-intensive arithmetic compared to the fast table lookup of 3-bit unums to populate 8-bit SORNs. With dedicated hardware for the low precision approach, the unum approach should reduce energy use and execution time by over a hundredfold, based on the number of exercised gates and the logic delay times.

If it is important to minimize the total number of constraint function evaluations, another approach is to split each of the twelve dimensions at a time, moving to the next dimension only when a split does not create a new excludable region. After six million low-precision calculations (requiring milliseconds to evaluate), the set of c_i-s_i pairs form arcs specific enough that a robotic control system would be able to make a decision, as shown in Fig. 7.

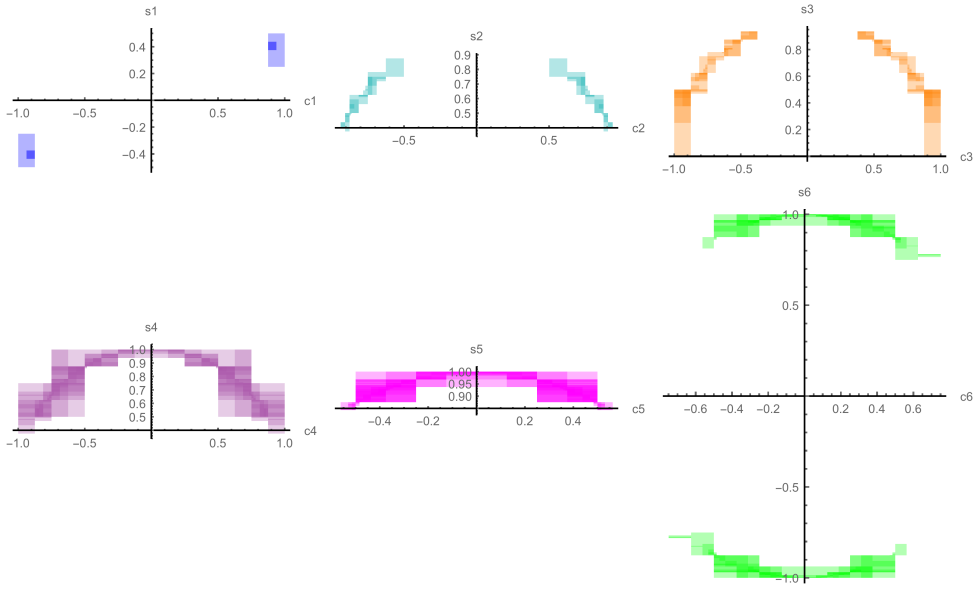


Figure 7. Proved feasible set for robotics inverse kinematics problem

4. Selecting the u-lattice and populating the number system

4.1. The u-lattice

Define the *u-lattice* as an ordered set of 1 followed by exact positive real numbers in the interval $(1, \infty)$. This set is then made closed under both negation and reciprocation, by including negatives and reciprocals of the u-lattice in the set of exact unums. For a fixed-size unum of length m bits, the u-lattice should have 2^{m-3} values, that is, 1/8 of 2^m, since 2^m is the total number of bit patterns possible with m bits. The reason is that combining the u-lattice with its reciprocal almost doubles the number of exact values (1 is already in the set), and combining with the negative of that set doubles it again; finally, representing the open intervals between exact values doubles the number of exact values a third time. Including the values 0 and $\pm\infty$ brings the total count up to exactly 2^m, so no bit patterns are wasted and no bit patterns are redundant.

4.2. Example for 4-bit unums

Four bits for each unum means $2^4 = 16$ bit patterns, and $2^{4-3} = 2$ values in the u-lattice. A simple example is to select $\{1,\ 2\}$ as the u-lattice. There is nothing special about the number 2; we could have used $\{1,\ 10\}$, or even $\{1,\ \pi\}$ as the exact numbers on which to base the number system. Some people are surprised that π can be represented as an exact *number*, but of course it can, which is one reason for the term "universal number."

The set $\{1,\ 2\}$ united with its reciprocals $\{1/2,\ 1\}$ becomes $\{1/2,\ 1,\ 2\}$. Uniting with negatives of the set and the set $\{0,\ \pm\infty\}$ gives the eight possible exact unums $\{\pm\infty,\ -2,\ -1,\ -1/2,\ 0,\ 1/2,\ 1,\ 2\}$. The last step is to include the open intervals between each of these, such as $(-1/2,\ 0)$, so there are also eight possible inexact unums, as shown in Fig. 8.

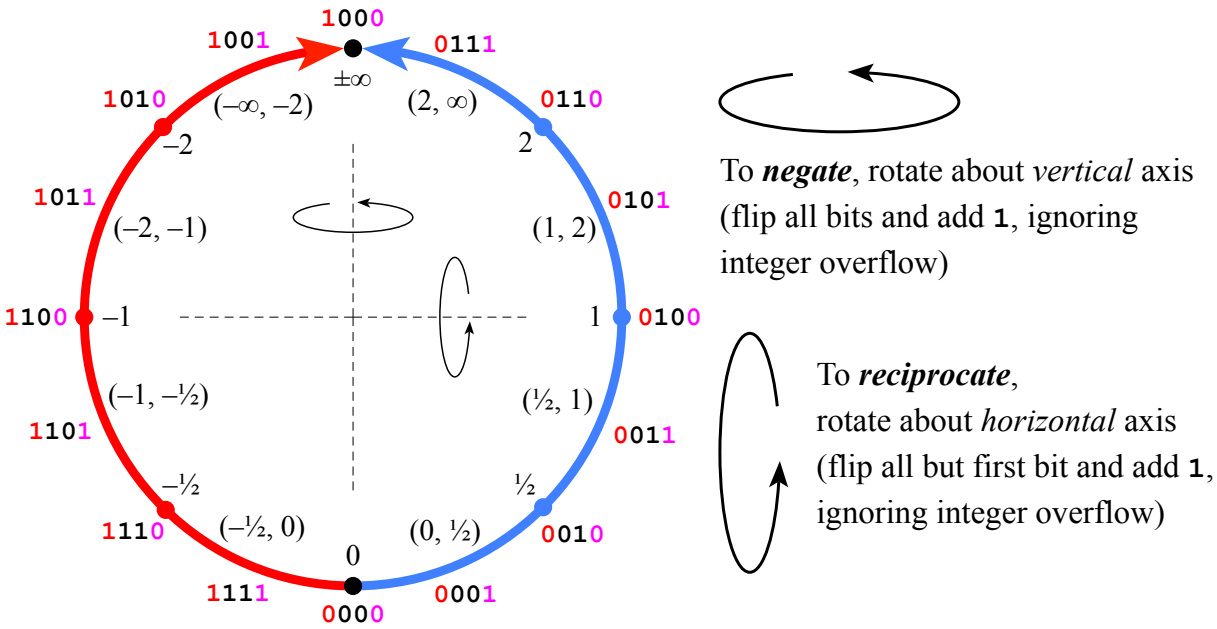


Figure 8. Four-bit unums and geometrical analogies for negating and reciprocating

This system places the arithmetic operations + − × ÷ on *equal footing*. With floats, it is dangerous to replace the operation $x \div y$ with $x \times (1/y)$ because there are two rounding operations; float math seldom calculates the reciprocal 1/y without rounding error, so for example, $3 \div 3$ may evaluate to 1 exactly, but $3 \times (1/3)$ will result in something like 0.9999···. On the other hand, it has long been safe to treat $x - y$ as identical to $x + (-y)$. Addition and subtraction share hardware. With the system described here, multiplication and division can share hardware as well.

It may be time to revive an old idea: "/" as a unary prefix operator. Just as unary "–" can be put before x to mean $0 - x$, unary "/" can be put before x to mean $1/x$. Pronounce it "over," so $/x$ would be pronounced "over x." Just as $-(-x) = x, //x = x$. Compiler writers and language designers certainly should be up to the task of parsing the unary "/" operator as they have with unary "–".

4.3. Freedom from division-by-zero hazards

What is $f(x) = 1/(1/x + 1/2)$ for $x = 0$? Most number systems balk at this and throw an exception because the $1/x$ step divides by zero. With the projective real approach used here, $1/0 = \pm\infty$; adding $1/2$ to $\pm\infty$ leaves $\pm\infty$ unchanged, and then the final reciprocal operation turns $\pm\infty$ back into zero. The expression can be rewritten as $f(x) = 2x/(2 + x)$, revealing that the singularity at $x = 0$ is perfectly removable, but that requires someone to do the algebra.

Suppose the input were a SORN, such as one representing the half-open interval $-1 < x \leq 2$. With the 4-bit unums defined in the previous section, the computation can be performed without any loss of information. The SORN sets remain contiguous sets through every operation, and provide the correct result, $-1/2 < f(x) \leq 1$, as shown in Fig. 9. The figure saves space by using the unary "/" notation, and we will use that notation from now on.

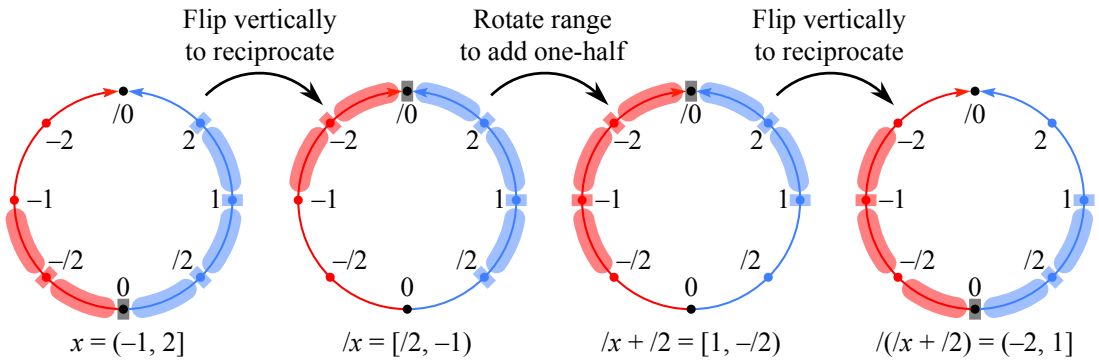


Figure 9. Tight bounding of $1/(1/x + 1/2)$ despite intermediate division by zero

Notice that if we had we rewritten the expression as $2x/(2+x)$ and attempted traditional interval arithmetic with $x = [-1, 2]$ (since we have no way to express open endpoints, we have to use a closed one at the -1 endpoint), $f(x)$ would evaluate to the loose bound $[-2, 4]$ because of the *dependency problem*: x now appears twice in the expression, and the calculation ignores the dependency between them. The interval arithmetic result thus unnecessarily includes the value -2 and the range $(1,\ 4]$, and is twice as wide a bound as the tight bound shown in Fig. 9.

4.4. Strategies for an 8-bit unum set

Since IEEE 754 specifies *decimal* floats only for 32-, 64-, and 128-bit sizes, it will be interesting to see if we can create a useful decimal system with as few as 8 bits. The choice of u-lattice depends on the application. If a large dynamic range is important, we could use this u-lattice: $\{1, 2, 5, 10, 20, 50, \ldots, 10^9, 2\times10^9\}$. The reciprocals of that set are also expressible with a single decimal times a power of 10, and that u-lattice provides over 18 orders of magnitude (from 5×10^{-10} to 2×10^9) of dynamic range, but less than one decimal digit of accuracy.

If we prefer to have every counting number from 1 to 10 represented, then we could start with the set $\{0.1, 0.2, \ldots, 0.9, 1, 2, \ldots, 9\}$ as "must have" values. This is what IEEE decimal floats do, and one of the drawbacks is "wobbling accuracy" when the slope changes suddenly. Deviation from a true exponential curve means that the relative error is too low in some places and too high in others, indicating information-inefficient use of bit patterns to represent real numbers. The left graph in Fig. 10 shows this effect, where the slope suddenly increases.

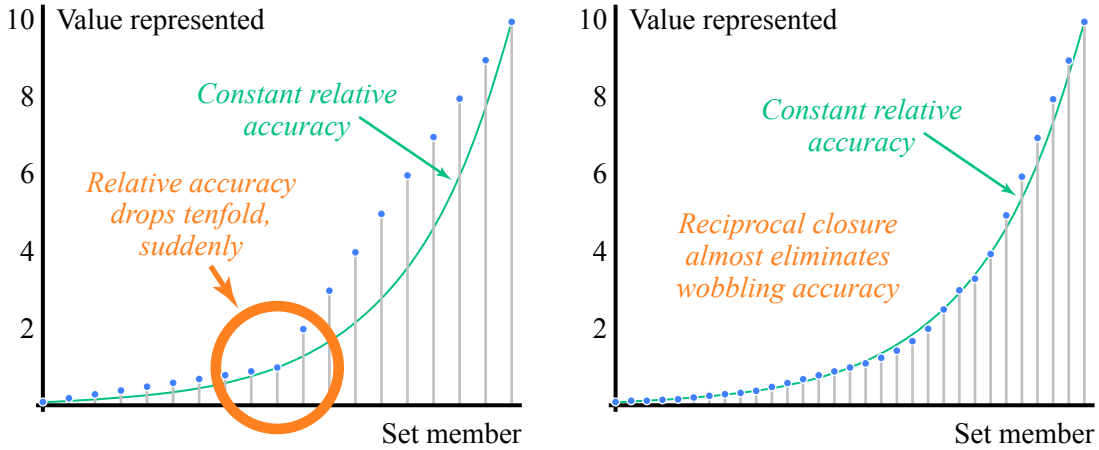


Figure 10. IEEE-style decimals versus decimal unums with reciprocal closure

Creating closure under reciprocation has the added benefit of dramatically reducing the wobbling accuracy in the selection of exact values. The width of uncertainty, divided by the magnitude of the value, is almost flat. The set of numbers may look peculiar since we are probably not yet used to reading the unary "/" operator, but from smallest to largest it sorts as follows:

$$\{0.1, /9, 0.125, /7, /6, 0.2, 0.25, 0.3, /3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,$$
$$1, /0.9, 1.25, /0.7, /0.6, 2, 2.5, 3, /0.3, 4, 5, 6, 7, 8, 9, 10\}$$

Wherever a reciprocal has an equivalent traditional finite decimal, we show the finite decimal; instead of writing "/8" for one-eighth, we write the familiar "0.125" for now, even though it takes five characters to express instead of two. This u-lattice has 15 exact values per decade of magnitude. That means an 8-bit unum lattice could range from 0.009 to /0.009, slightly more than four orders of magnitude. Compared to the {1, 2, 5, 10, ...} u-lattice, we have less dynamic range but a solid decimal of accuracy while preserving closure under reciprocation.

There is an intermediate between these u-lattices that merits attention, even if it has no obvious way to scale to more decimals of accuracy: Powers of 2 ranging from −4 to +4, scaled to fit into the 1 to 10 range. That is, start with the following set:

$$\{0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, 16\},$$

then scale each entry to fit into the 1 to 10 range:

$$\{1, 1.25, 1.6, 2, 2.5, 4, 5, 6.25, 8, 10\}.$$

This is a wonderful basis for a u-lattice, for those concerned with closure under multiplication and division. It looks like a nearly exponential spacing of points from 1 to 10, except for the relatively large gap between 2.5 and 4. Plug that gap with $\sqrt{10}$ and an amazing thing happens: the u-lattice produces a very close match to ten exponentially spaced points, as shown in tab. 3. Engineers will recognize the bottom row as the definition of *decibel* ratios, from 0 dB to 10 dB.

Table 3. A new way to count from 1 to 10

Exact unum	1	1.25	1.6	2	2.5	$\sqrt{10}$	4	5	6.25	8	10
$10^{k/10}$, k = 0 to 10	1	1.25···	1.58···	1.99···	2.51···	$\sqrt{10}$	3.98···	5.01···	6.30···	7.94···	10

By crafting the u-lattice this way, we obtain even less wobbling accuracy than binary floats, for which relative accuracy wobbles by a factor of 2. Some may balk at $\sqrt{10}$ being treated as a counting number, if for no other reason than it being difficult to type, but if written as "r10" then it should present no problem for computer input as a character string. The set of positive exact unum values gives over six orders of magnitude:

$$\{0.0008, 0.001, 0.00125, 0.002, 0.0025, 0.001r10, 0.004, \ldots, 100r10, 400, 500, 625, 800, 1250\}.$$

It is worth looking at the multiplication and addition tables for a decade's worth of values, to see how often a result is expressible as another exact unum (cyan) versus lying between two exact unums (red). The table leaves out input arguments 1 and 10 as trivial, and we write "···" after

an exact unum as the shorthand for "in the open interval beyond this exact unum," indicating information loss.

Table 4. Multiplication table within a decade

×	1.25	1.6	2	2.5	r10	4	5	6.25	8
1.25	1.25···	2	2.5	2.5···	r10···	5	6.25	6.25···	10
1.6		2.5···	r10···	4	5···	6.25···	8	10	12.5···
2			4	5	6.25···	8	10	12.5	16
2.5				6.25	6.25···	10	12.5	12.5···	20
r10					10	12.5···	12.5···	16···	25···
4						16	20	25	10r10···
5							25	25···	40
6.25								10r10···	50
8									62.5···

A remarkable 25 of the 45 entries shown are exact. A desirable property of a u-lattice is that there not be too much "blurring" of results from the basic operations. The product of an exact unum and an inexact one should not require more than three contiguous unums: an inexact, an exact, and an inexact. The product of two inexact unums should not spread out to more than five contiguous unums. Having a nearly exponential spacing of u-lattice values helps achieve this goal.

There is obvious symmetry about the diagonal from upper left to lower right, since multiplication is commutative. There is a less-obvious symmetry about the orange cells shown from the top right to the center of the table; numbers reflected over those cells become the reciprocal (times 10) of the value, since we have closure under reciprocation. We could omit those, just as the table omits negative values. While a naïve table of all possible 128 by 128 multiplications (16 384 entries) could be maintained if memory is cheap and logic expensive, a little bit of logic can go a long way to reduce the number of necessary table entries to as few as 45.

4.5. Other strategies worth considering

Rational arithmetic has some advocates, where the numbers are of the form $\pm p/q$ where p and q are positive integers up to some limit, together with some accommodation for zero and infinity cases. (Without ubit support, rational arithmetic suffers the same rounding problems as floats since most operation results will not be exactly expressible.) The u-lattice provides excellent scaffolding for the creation of a rational number system. For example, we could find all p/q such that $1 \le q \le p \le 10$ to populate one decade of a u-lattice:

$$1, 10/9, 9/8, 8/7, 7/6, 6/5, 5/4, 9/7, 4/3, 7/5, 10/7, 3/2, 8/5, 5/3,$$
$$7/4, 9/5, 2, 9/4, 7/3, 5/2, 8/3, 3, 10/3, 7/2, 4, 9/2, 5, 6, 7, 8, 9, 10$$

This could be simplified by requiring that $p + q \le 11$:

$$1, 6/5, 5/4, 4/3, 3/2, 5/3, 7/4, 2, 7/3, 5/2, 3, 7/2, 4, 9/2, 5, 6, 7, 8, 9, 10$$

In either case, it is easy to obtain closure under reciprocation, and a fair number of multiplications and divisions land on exact values. The percentage of the time an exact result is

produced, however, is not as high as it is for the "decibel" number set, and because there are twice as many exact values per decade, the dynamic range will be half as large. However, if an application demands many ratios of small integers, this approach may have its uses.

Finally, we have been showing *flat accuracy* systems where the spacing of the logarithm of values is approximately constant from smallest to largest positive lattice point. Another approach is to use *tapered accuracy* where there is more accuracy near 1 but less accuracy for very large and very small values. For example, with 8-bit unums we could still populate the decade between 1 and 10 as shown in the right-hand graph of Fig. 10:

$$1, /0.9, 1.25, /0.7, /0.6, 2, 2.5, 3, /0.3, 4, 5, 6, 7, 8, 9,$$

but then increase the spacing for the next decade:

$$10, 12.5, 20, 25, 40, 50, 80,$$

still wider spacing in the next decade,

$$100, 200, 500,$$

and finally allow the exponent to grow so rapidly that it becomes very unlikely for a product to land in the (maxreal, /0) open interval:

$$1000, 10\,000, 10^6, 10^{10}, 10^{20}, 10^{50}, 10^{100}.$$

These 32 exact values, united with their reciprocals, negatives, 0 and /0, and the open intervals between those values, form a byte-sized unum that is decimal-based with slightly more than one digit of accuracy near unity and a dynamic range of 200 orders of magnitude. The main drawback to tapered accuracy is that it is harder to compress the look-up tables by exploiting patterns that repeat for every decade.

5. Why unums are *not* like interval arithmetic

Perhaps the most succinct form of the interval arithmetic "dependency problem" is this: Assign x to an interval, and compute $x - x$. Obviously the correct answer is zero, but that's not what interval arithmetic gives you. Suppose x is the interval [2, 4]. If we assign $x \leftarrow x - x$ repeatedly you will get the following interval ranges after a few iterations:

$x = [-2, 2]$
$x = [-4, 4]$
$x = [-8, 8]$
$x = [-16, 16]$
$x = [-32, 32]$

The bounds grow exponentially. The interval method takes $(\max x) - (\min x)$ for the upper bound and $(\min x) - (\max x)$ for the lower bound. The uncertainty feeds on itself. In contrast, a SORN with the 8-bit unums shown in tab. 3 produces the following sequence:

$x = (-1, 1)$ $\{(-1, -0.8), -0.8, (-0.8, -0.625), \ldots, (0.625, 0.8), 0.8, (0.8, 1)\}$
$x = (-0.2, 0.2)$ $\{(-0.2, -0.16), -0.16, \ldots, 0.16, (0.16, 0.2)\}$
...
$x = (-0.0008, 0.0008)$ $\{(-0.0008, 0), 0, (0, 0.0008)\}$

The sequence actually *decreases* in width and is stable. Similarly, $x \leftarrow x/x$ blows up very rapidly if iterated similarly using interval arithmetic:

$x = [1/2, 2]$
$x = [1/4, 4]$
$x = [1/16, 16]$
$x = [1/256, 256]$
$x = [1/65\,536, 65\,536]$

whereas the SORN set converges to a stable interval containing 1, with some loss of information caused by the limited (single-digit) accuracy. Because arguments to arithmetic operators are never more than one ULP wide, the expansion of the bounds cannot feed on itself. An n-body simulation written using unums shows only linear growth in the bounds of the positions and velocities, but interval arithmetic quickly produces bounds of meaningless large size [2].

6. Higher precision, and table look-up issues

6.1. 16-bit unums and a comparison with half-precision floats

As we increase the unum size to 16 bits and larger, we start to notice the need for techniques to reduce the size of SORN representation and the size of look-up tables. If we start with a contiguous set of unums and only use $+ \; - \; \times \; \div$ operations, the unums *remain contiguous*. This property means that for an n-bit unum, the SORN can be represented with two n-bit integers, the first indicating the position of the first **1** bit and the second indicating the length of the string of **1** values. The pair of integers 0, 0 is reserved to represent the empty set, and any other identical pair is reserved to represent a SORN with all presence bits set, that is, $[-/0, /0]$ or $\mathbb{R}^+$.

IEEE half-precision binary floats have slightly more than 3-decimal accuracy, and the normalized numbers range from about 6×10^{-5} to 6×10^4, or nine orders of magnitude dynamic range. Many bit patterns are wasted on redundant NaN representations and negative zero. Can a 16-bit unum do as well, and actually *express three-digit decimals exactly*?

The surprising answer is that 16-bit unums cover *more* than nine orders of magnitude, from $/0.389 \times 10^{-5}$ to 3.89×10^4, despite the cost of reciprocal closure and the ubit to track inexact results. Sometimes, an answer known accurate to three decimals is preferable to a (64-bit) 15-decimal answer of completely unknown accuracy. It is also possible to represent all the 2-decimal values with 16-bit unums, which allows a dynamic range of more than *93* orders of magnitude.

In general, to store decimals from $1.00\ldots0$ (k digits) to $9.99\ldots9$ (k digits) requires almost 3.6×10^k distinct unum values, including the exact reciprocals and the values between exact unums. For example, a decimal unum equivalent to an IEEE 32-bit binary float might have 7 decimal digits of accuracy, and over 59 orders of magnitude dynamic range. If representation of physical constants like Avogadro's number and Planck's constant are important, we could settle for 6 decimals of accuracy and then be able to represent almost 600 orders of magnitude.

6.2. Table look-up issues and future directions

Arithmetic tables are rich in symmetries that can be used to reduce their size, at the cost of some conditional tests and some integer divides. For example, do we need both positive and negative entries in the multiplication table, or should we test the sign of each argument, make them positive if necessary, and then set the sign as appropriate? That tiny bit of logic cuts the table size by a factor of four, yet for extremely low-precision unums it may not be worth it! As the unum precision increases, we find ourselves wanting more logic and smaller tables.

Suppose we did a very naïve set of tables for $+ \; - \; \times \; \div$ for 16-bit unums, by fully populating all four tables and ignoring symmetry and repeating patterns. That would require 32 gigabytes. But a small amount of obvious logic easily reduces that to a few megabytes. A more sophisticated approach notices that the smoothly exponential spacing of a flat-accuracy u-lattice means that the unum bit strings can be added or subtracted to perform a close approximation to multiplication or division, and the table need only hold small integer correction factors.

The next research direction is to find a practical set of techniques for fast table look-up and for table size minimization, and determine tradeoffs between speed and table size. While the table sizes for higher precision unums may look daunting, it is possible to put many billions of bits of ROM storage in an integrated circuit without ruining the power budget or the area demands. It may turn out that the new formulation of unums is primarily practical for low-accuracy but high-validity applications, thereby complementing float arithmetic instead of replacing it.

7. Summary: Software-Defined Arithmetic

The approach described here could be called *software-defined arithmetic* since it can adjust to the needs of a particular application. In fact, an application could alter its number system from one stage to the next by pointing to alternative look-up tables; perhaps the early stages of a calculation must test a large dynamic range, but once the right range is found, bits in the representation are put to better use improving accuracy. Little is needed in the way of specialized hardware for "Type 2 unums" since all processors are well equipped with the machine instructions look up values in tables. Perhaps a processor could dedicate ROM to a standard set of u-lattice values, but allow the user to define other u-lattices in RAM, at the cost of slightly more energy and execution time. The processor instructions for processing SORNs would not need to change.

The *compiler* could select the u-lattice, especially if the compiler has information (provided by the user or by historical data from the application runs) regarding the maximum dynamic range needed, say, or the amount of accuracy needed. Perhaps the greatest savings in time would be for the compiler to populate look-up tables for *unary* functions needed in any particular program. Imagine a program that repeatedly computes, say, an expression like $\cos\left(x + e^{-3x^2}\right)$. The compiler could discover the common sub-expression, then provide a table that looks up the value in only one clock cycle and is mathematically perfect to the unum accuracy level. (The "Table Maker's Dilemma" disappears since there is no requirement of constant-time evaluation at run time.)

The energy savings and speed of such customized arithmetic might well be one or two orders of magnitude better than IEEE floats, while providing automatic control of accuracy loss in a way that resists the shortcomings of interval arithmetic. Type 2 unums may well provide a shortcut to achieving exascale computing.

*This work was supported by the A*STAR Computational Resource Centre.*

References

1. Yousif Ismaill Al-Mashhadany. Inverse kinematics problem (IKP) of 6-DOF Manipulator by Locally Recurrent Neural Networks (LRNNs). In *Management and Service Science (MASS), 2010 International Conference on*, pages 1–5. IEEE, 2010.

2. John L. Gustafson. *The End of Error: Unum Computing.* CRC Press, 2015.

3. Timothy Hickey, Qun Ju, and Maarten H. Van Emden. Interval arithmetic: From principles to implementation. *Journal of the ACM (JACM)*, 48(5):1038–1068, 2001.

4. William M. Kahan. A more complete interval arithmetic. *Lecture notes for a summer course at the University of Michigan*, 1968.

5. Jean-Michel Muller, Nicolas Brisebarre, Florent De Dinechin, Claude-Pierre Jeannerod, Vincent Lefevre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of floating-point arithmetic.* Springer Science & Business Media, 2009.

6. Dan Zuras, Mike Cowlishaw, Alex Aiken, Matthew Applegate, David Bailey, Steve Bass, Dileep Bhandarkar, Mahesh Bhat, David Bindel, Sylvie Boldo, et al. IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.

DOI: 10.14529/jsfi160204

InfiniCloud 2.0: Distributing High Performance Computing across Continents

Jakub Chrzeszczyk[1]***, Andrew Howard***[1]***, Andrzej Chrzeszczyk***[2]***, Ben Swift***[1]***,***
Peter Davis[1] ***Jonathan Low***[3]***, Tin Wee Tan***[45]***, Kenneth Ban***[5]

InfiniCloud 2.0 is the world's first native InfiniBand High Performance Cloud distributed across four continents, spanning Asia, Australia, Europe and North America. The project provides researchers with instant access to computational, storage and network resources distributed around the globe. These resources are then used to build a geographically distributed, virtual supercomputer, complete with globally-accessible parallel file system and job scheduling. This paper describes the high level design and the implementation details of InfiniCloud 2.0. Two example applications types, a gene sequencing pipeline and plasma physics simulation code were chosen to demonstrate the system's capabilities.

Introduction

The original InfiniCloud system, presented at Supercomputing Frontiers Singapore in March 2015, enabled researchers to quickly and efficiently copy large volumes of data between Singapore and Australia, as well as to process that data using two discrete, native InfiniBand High Performance Clouds [8]. It also provided an opportunity to establish a detailed baseline of compute, memory, storage and network performance of native Infiniband High Performance Cloud [11].

While the unique capabilities of InfiniCloud enabled new ways of processing data, it also inspired a whole new range of research questions: Can the entire capacity of the system be aggregated? Do entire data collections need to be copied for processing (even with a 99% effective circuit efficiency delivered using extended InfiniBand), or can data be accessed in place? How does the InfiniCloud design scale to an arbitrary number of sites? How we ensure a consistent state of all InfiniCloud clusters? And finally, can the resources across four continents be joined together using the InfiniCortex fabric to create a Galaxy of Supercomputers [14]?

In this paper we aim to explore these research questions and propose new ways of utilizing distributed computation, storage and network resources, using a variety of novel tools and techniques. We aim to provide a unified interface allowing users to transparently access resources at each and every site using a standardized set of CLI, GUI and API tools. We also take advantage of the expansion and enhancement of the InfiniCortex fabric which took place in 2015 [12], which includes full support for InfiniBand subnets and routing, greater available bandwidth and last but not least the growing number of participating sites. Finally, we demonstrate new and unique capabilities of the system by deploying example scientific applications across geographically distant sites.

[1]The Australian National University
[2]Jan Kochanowski University, Kielce, Poland
[3]A*STAR Computational Resource Centre (ACRC), Singapore
[4]National Supercomputing Centre (NSCC), Singapore
[5]Dept. of Biochemistry, Yong Loo Lin School of Medicine, National University of Singapore

1. The Network

The InfiniCortex network, which is crucial to the existence of InfiniCloud, has expanded from an experimental service connecting Australia and Singapore, through a cycle of significant extension, improvement in operational stability and enhancement of fabric isolation throughout 2015 to create for a time an InfiniBand Ring Around the World.

The underlying physical network is constructed using interconnected Advanced Layer 2 Services (AL2S) provided by National Research and Education networks to deliver a set of loss-less 10Gbs Layer 2 channels reaching around the globe. Each presenting a fixed latency and data rate. To date, the network has been instantiated on-demand to support ISC and SC, during 2016 a permanent InfiniCortex core will be created linking Europe, Asia and the Americas.

A significant characteristic of the InfiniCortex network is the optimised presentation of both high bandwidth and fixed latency using InfiniBand as the underlying transport protocol. When compared to traditional IPv4 as the transport protocol the deterministic nature of InfiniBand delivers a direct improvement of large data set transfer data rates. While we have demonstrated this capability at an international scale we believe that it provides greater advantages at a Campus, Metro or Regional scale with lower inter-site latency to scale transparently beyond a single data centre or facility. (Figure 1, Figure 2)

1.1. Connecting to Europe and additional US based facilities

The most significant change to the InfiniCortex in 2015 was the establishment of native InfiniBand connectivity to Europe, thanks to help and support from many National Research and Education Networks including AARNet, Internet2, TEIN*CC, SingAREN and Geant. This enabled InfiniCortex sites to connect to Warsaw and Poznan in Poland as well as Reims in France, allowing the University of Reims to become a new InfiniCloud node (Figure. 2). Connectivity to the East Coast USA has been maintained and also been significantly enhanced, enabling Stony Brook University in New York to also join InfiniCloud project.

1.2. Transition to 100Gbit/s Networks and InfiniBand

The second significant change to the InfiniCortex 2015 was the phase change from 10G and 40G to 100G as an advanced research network service. This allowed additional virtual circuits to be created between Asia and sites located in North America, allowing higher data transfer performance, supporting a greater level of concurrency for the increased volume of network traffic.

We anticipate in 2016 as 100G network capabilities become more prevalent and affordable and the implementation of 100G EDR based InfiniBand fabrics more widespread, that the capabilities of the InfiniBand Extension and Routing devices underlying the InfiniCortex will follow, allowing the removal of the current bandwidth impedance mismatch of a 10G network connecting FDR 56G or EDR 100G InfiniBand fabrics and supporting line rate 100G inter-fabric communication in the near future.

This will provide the opportunity to exploit the deterministic nature of data access in RDMA capable applications operating in this environment. We have begun exploring the use of a distributed file system layer which is tuneable for the InfiniCortex network characteristics. Allowing a change in the conventional paradigm of staging large data set into and out of a remote facility for processing to one of data access in place using the underlying opportunistic caching

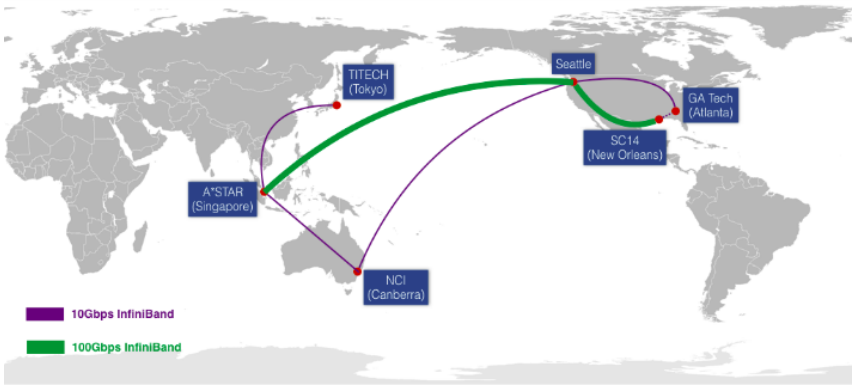


Figure 1. InfiniCortex 2014 network diagram

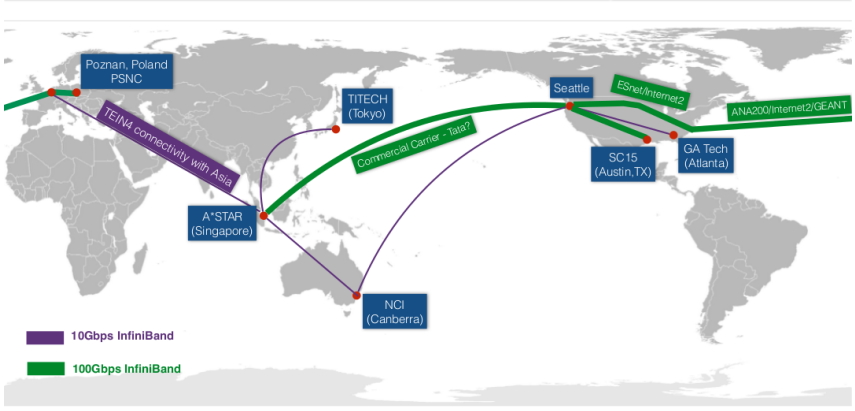


Figure 2. InfiniCortex 2015 network diagram.

capability of the filesystem to optimise inter-facility traffic. Our initial investigation has shown that gene alignment, sequencing and annotation applications with a primarily serial data access pattern perform well in this environment. Analysis and characterisation of the performance of additional classes of applications at both a metro scale and global scale environment are underway.

1.3. Routable InfiniBand

The third key change was transition from a single, centrally managed InfiniBand fabric to a interconnected pool of independent InfiniBand subnets, connecting to each other using the Obsidian Strategics R400 InfiniBand Router. This transition has significantly enhanced InfiniCortex operational stability - by eliminating crosstalk between local subnets, reducing latency in Subnet Manager communication by orders of magnitude and containing any network issues within a single site, minimizing the impact on other sites. All this was achieved without any impact on network performance and the ability to efficiently and securely transfer data. All the key components of the InfiniCortex network implementation are listed in (Table 1)

Progressive enhancements and improvements to the InfiniCortex have laid a foundation for building the next generation of geographically distributed High Performance Computing and Clouds. As the network matures from an experimental platform to a operational service, ongoing performance characterisation is being undertaken and the findings will be presented in a future paper, the main focus of this paper however is on the Cloud infrastructure.

Table 1. InfiniCortex long-range InfiniBand equipment

InfiniBand extenders	Obsidian Strategics Longbow^{TM} E100
InfiniBand routers	Obsidian Strategics Crossbow^{TM} R400
Layer 2 connectivity	Multiple loss-less, QoS-enabled 10GE circuits

1.4. Private, software-defined networks

With the latest versions of the OpenStack and Obsidian products, it is now fully supported to implement a number of private, secure, software-defined InfiniBand networks. Each project described in this paper was contained within a dedicated InfiniBand subnet, with the option of reliable, secure, high-performance external connectivity if required.

2. The Cloud

The founding requirements of InfiniCloud 2.0 were i) simplifying and standardizing access to the resources provided by the distributed system, ii) enabling new ways of using the network capabilities for interacting with data iii) enabling easy scaling to more than two sites.

2.1. InfiniCloud rationale and the existing solutions

The original motivation behind exploring the novel topics of long-range, routable InfiniBand, as well as connecting it to the Cloud was solving one of the ultimate challanges underlying Cloud Computing - the ability to work with large volumes of data in a geographically distributed context. While Public Cloud resources are available on demand and at low cost to users anywhere in the World, the easy access to computational and storage resources becomes much more difficult as soon as we add the requirement of accessing large quantities of data from outside any of the popular Public Clouds.

One issue with Public Cloud data transfer is performance - from our experience, copying a typical genomics sample of 300GB out of a Cloud hosted on the East Coast US to Singapore can take as long as 24 hours. This itself makes it impossible to enable real-time aggregation of compute and storage to achieve HPC-grade performance - where the boundary is previous-generation interconnect performance - 10Gbit/s or 1GB/s. Also significant the cost of data transfer out of Public Clouds is several orders of magnitude higher than the cost of compute within those Clouds.

To provide an example - Amazon EC2 instances compute instances are priced from 0.01$/hr for t2.micro to 1.68$/hr for c3.8xlarge. At the same time, data transfer rates are from 0.02$/GB for inter-region transfers within Amazon to 0.09$/GB for data transfers to the Internet [1]. With these rates, even if we were to hypothetically run transfers at the speed we're aiming for (1GB/s) which we weren't able to achieve, this would amount to 72$/hr for inter-region transfers or to 324$/hr for transfers to the Internet.

In any possible case, the comparison of the cost of compute and network traffic (out by two orders of magnitude) clearly states that the use case discussed in this paper is not possible to implement efficiently using today's Public Cloud model.

The use cases typical to most cloud users do not involve moving such huge volumes of data on a regular basis hence this is not a major focus for the Public Cloud operators. However, from

our experience, organizations such as genomic consortium's and large multi-national companies do have such requirements.

The advantage that InfiniCloud 2.0 is proposing is a mechanism which can enable virtual aggregation of geographically distant data centres - in a secure and energy efficient way utilising the next generation of metro, regional and global networks. This is done by connecting Cloud to a global-range, routable Infiniband fabric for the first time.

As no such capability is currently available from any of the Public Cloud providers, we envision that just like dial-up was replaced by broadband links supporting unlimited data transfer, inefficient and expensive data transfer rates charged per gigabyte will ultimately be replaced with solutions similar to those explored by this collaboration using Research and Education networks and described in this paper.

2.2. Cloud Architecture

In order to fulfill some of these requirements, a significant re-design of the system was required. The underlying model of independently operated, site-contained clusters which leverage the InfiniCortex fabric for high performance, efficient data transfer between cloud instances was replaced with a more tightly coupled, experimental architecture, utilising a central Cloud Controller with Compute Nodes distributed around the globe.

This change implemented the first requirement: a uniform user interface with a single CLI / GUI / API interface accessible to users. It also removed the necessity to manually synchronize state between different clusters which was a requirement in the previous, loosely-coupled design.

With the architecture described above, it is crucial to carefully consider the characteristics of the network, especially in terms of latency, in order to make optimal component placement decisions. It is highly recommended that consideration of placement of the critical components (Cloud Controller, Subnet Manager, and at later stage - also parallel file system instances and the job scheduling instance) occurs in a way to ensure that the latency between all nodes is consistent.

In the scenario presented in this paper, Singapore provided the best location for these services. A consistent, 300ms round-trip latency from Canberra, Reims and New York as well as access to 100Gbit/s connectivity made the South-East Asian city-state the optimal location for the Cloud Controller and the Master Subnet Manager.

Two distinct pools of compute nodes were also hosted in Singapore, joined by more nodes hosted in Canberra, New York and Reims, all connected with high performance RDMA and IPoIB connectivity provided by the InfiniCortex.

The last but not the least, given the nature of the system, users needed to be able to have fine-grained control over scheduling of the resources requested, be it compute or storage. Some applications might be suitable to be distributed around the world in a round-robin fashion, while others may require user-defined, role, facility, regional, sovereign territory or geographically based scheduling.

2.3. Cloud implementation

All InfiniCloud systems run the following hardware (Table 3) and software stack (Table 2).

The deployment of the system begins with building the Cloud Controller in Singapore.

Table 2. InfiniCloud 2.0 software stack

Operating System	CentOS 7.1 x86_64
InfiniBand drivers	Mellanox OFED
OpenStack version	Kilo (customized)

Table 3. InfiniCloud 2.0 hardware configuration

CPUs	Intel Haswell (Singapore), Sandy Bridge and Ivy Bridge (other sites)
Memory	64GB-256GB
Interconnect	Mellanox FDR
Local storage	Intel DCS family SSDs (Singapore), SAS HDDs (other sites)

2.3.1. Cloud Controller

Due to the complexity of this component, it is performed in a multi-step process, starting with a kickstart build and then switching to a puppet configuration management solution based on widely adopted PuppetForge OpenStack modules [5] [6] . After the base installation is completed, the custom, out-of-tree InfiniCloud2.0 specific patches are applied, which completes the Cloud Controller installation.

2.3.2. Compute Nodes

InfiniCloud 2.0 Compute nodes are much simpler than the controller node and as such can be kickstart-built. As soon as the freshly-built nodes report to the Cloud controller using the InfiniCortex fabric, their configuration is pushed and they become ready for operation.

Two compute nodes are build in each of the InfiniCloud 2.0 locations. Additional capacity is available in Singapore for infrastructure services, such as a virtual parallel file system.

2.4. Resource scheduling

The default behaviour of the OpenStack scheduler is to allocate virtual instances to compute nodes in a round robin fashion. While this is a reasonable approach for a homogeneous cluster hosted in one location, it is unsuitable for a globally distributed system, where - as discussed in the previous sections - the locations of particular components have to be carefully optimized for consistent latency.

2.4.1. Availability zones

For the above reason, a separate availability zone was created for each location. Each zone (Australia, Asia, Europe, USA) corresponds to a physical location (Canberra, Reims, Singapore, New York) and holds all the compute resources that are hosted in this location.

On top of these, two additional availability zones were added: local and distributed. The local zone is reserved for infrastructure use (such as Sun Grid Engine head node or BeeGFS parallel file system nodes) and instances allocated to this zone will always be launched in Singapore. The distributed zone stripes across all the locations and provides a simple mechanism to distribute workloads across the remote nodes. This is illustrated in (Listing 2.4.1 and Listing 1). **Remark:** For the purpose of this paper and for clarity let us assume that host aggregates and availability zones are synonyms.

```
# availability zone example
[root@ics111 ~(keystone_admin)]# nova aggregate-list
+----+-------------+---------------------+
| Id | Name        | AvailabilityZone    |
+----+-------------+---------------------+
| 1  | local       | -                   |
| 7  | distributed | -                   |
| 8  | singapore   | singapore           |
| 9  | australia   | australia           |
| 10 | europe      | europe              |
| 11 | america     | america             |
+----+-------------+---------------------+
```

Listing 1. InfiniCloud 2.0 availability zones

```
# zone to host mapping example
[root@ics111 ~(keystone_admin)]# nova aggregate-details 7
+----+-------------+-----------------------------------+-------------------+
| Id | Name        | Hosts                             | Metadata          |
+----+-------------+-----------------------------------+-------------------+
| 7  | distributed |'ica120.infinicloud.nci.org.au',   | 'distributed=true'|
|    |             |'ica121.infinicloud.nci.org.au',   |                   |
|    |             |'ics130.infinicloud.a-star.edu.sg', |                   |
|    |             |'ics131.infinicloud.a-star.edu.sg', |                   |
|    |             |'icu143.infinicloud.stonybrook.edu',|                   |
|    |             |'icu144.infinicloud.stonybrook.edu',|                   |
|    |             |'icf157.infinicloud.univ-reims.fr', |                   |
|    |             |'icf158.infinicloud.univ-reims.fr'  |                   |
+----+-------------+-----------------------------------+-------------------+
```

Listing 2. InfiniCloud 2.0 host to availability zone mappings

2.4.2. Instance types

While launching Cloud instances, the user has an ability to explicitly specify the intended availability zone. However, to enable a high degree of automation, it is recommended to create dedicated instance types which are linked to a particular availability zone and/or a scheduling pattern. This can be implemented using instance type metadata. This is illustrated in Listing 5. After the metadata is set, when instance types such as "local.2c.8m.170d" (local storage) or "geo.8c16m20d" (distributed compute) are requested, instances will be scheduled in the corresponding locations. These instance types can be referenced directly in applications such as ElastiCluster which is described in the next section. Examples of creating instances in a specific location or across all locations is demonstrated in Listing 3 and Listing 4.

```
[root@ics111 ~(keystone_admin)]# nova boot --flavor geo.2c2m20d --key-name oskey
--image BioPipeline_v0.8.6.7 --num-instances=4
--availability-zone singapore|australia|america|france singlezone
```

Listing 3. Creating cloud instances on a particular continent

```
[root@ics111 ~(keystone_admin)]# nova boot --flavor geo.2c2m20d --key-name oskey
--image BioPipeline_v0.8.6.7 --num-instances=4 distributed
```

Listing 4. Creating cloud instances across all continents

```
# instance types
[root@ics111 ~(keystone_admin)]# nova flavour-list --all   extra-specs
+---------+---------------+------+----+---------+----+------+---------------+
| ID      | Name          |Memory|Disk|VCPU|Public|extra_specs                |
+---------+---------------+------+----+----+------+--------------------------+
| 0e...db | local.2c8m170d| 8192 | 20 | 2  | True |{u'local': u'true'}       |
| 3a...5e | geo.8c16m20d  |16384 | 20 | 8  | True |{u'distributed': u'true'}|
| 60...7f | local.2c8m120d| 8192 | 20 | 2  | True |{u'local': u'true'}       |
| 83...52 | geo.4c8m20d   | 8192 | 20 | 4  | True |{u'distributed': u'true'}|
+---------+---------------+------+----+----+------+--------------------------+
```

Listing 5. InfiniCloud2.0 host to availability zone mappings

2.5. Communication patterns

There are three main types of network traffic which are relevant to operating a geographically distributed OpenStack cluster: Message queue traffic (AMQP), cloud endpoint traffic (HTTP), and downloading images (HTTP). All these protocols are native to IP, so IPoIB is used to carry the relevant traffic.

2.5.1. Bandwidth and latency considerations

Both message queue and endpoint traffic is only minimally bandwidth intensive and tolerant to large latency, so no tuning was required. In the case of images, while underlying HTTP protocol is robust and built-in auto-tuning mechanisms are useful, it is not able to fully utilize the available bandwidth over a 10Gbit/s connection with a 300ms round-trip latency. While the default configuration is usable, initial image caching can take several minutes (in case of the processing method described in this paper, this is a one off operation, so this delay is acceptable). This could be addressed with additional network tuning or the use of a custom image transport mechanism, ideally using native RDMA communications. Implementing such a mechanism, however is outside of the scope of this paper.
The amount of bandwidth consumed by the Cloud itself is minimal, and nearly all the bandwidth can be used by the applications running in virtual instances.

3. The Applications

This section will cover a selection of example applications that can take full advantage of the geographically distributed, High Performance Cloud: BeeGFS, Geopipeline and Extempore.

3.1. BeeGFS

One of the key advantages of high bandwidth, RDMA-capable interconnect spanning across continents is providing the ability to efficiently work with data. In our previous work, this mainly meant high speed data transfers and/or data synchronization. However, in many cases it might be more efficient to access datasets in-place, without the need to move parts or even the entirety of the data set. In this experiment, we use a BeeGFS parallel filesystem cluster hosted in Singapore to export data to consumers in Australia, Europe and the US [2].

The BeeGFS cluster used in this experiment consisted of three nodes. The first node was running management *(fhgfs-mgmtd)*, metadata *(fhgfs-meta)* and storage *(fhgfs-storage)* processes. The other two nodes were running solely the *fhgfs-storage* process. The storage space was presented as a qcow2 300GB ephemeral drive which was stored on a a single SSD drive. BeeGFS performance is typically determined by backing store read/write bandwidth - in our case 3xSSD drives, each capable of 500MB/s read/write, provided a 1500MB/s theoretical maximum I/O capability and given some overhead in the parallel filesystem layer, 1000MB/s is a typical value delivered. The network bandwidth needs to match or exceed the storage bandwidth. In our case, 56Gbit/s is available to local clients and 10Gbit/s is available to remote clients so this requirement is met. CPU performance and memory allocation have less impact on BeeGFS performance, however BeeGFS cluster under load manifests medium CPU utilization and sufficient memory is essential to support high-latency of an intercontinental link due to data buffering. We used 2 CPU cores and 8GB of RAM per server. All data transfer communications use native RDMA.
While mounting a file system across a link with such a large Bandwidth Delay Product (BDP) brings a number of technical challenges, we were able to derive the tuning parameters optimized for such scenario.

3.1.1. BeeGFS configuration for high bandwidth-delay products

This cluster was then specifically tuned for a large Bandwidth Delay Product inherent in high latency links. The key configuration value are included in (Listing 3.1.1). An important remark is that this results in an increased memory utilization, so the virtual instances need to be created with sufficient memory allocation to support this requirement. The default client time-out values were also increased by an order of magnitude.

```
> connRDMABufSize                 = 8192
> connRDMABufNum                  = 128

< connRDMABufSize                 = 65536
< connRDMABufNum                  = 260
```

Listing 6. BeeGFS tuning parameters

3.1.2. Optimizing data access patterns

In order to be able to obtain performance near or matching the local performance, the I/O patterns in use also need to be optimized. We aim to optimize data access methods to ensure as much data remains in flight as possible and to ensure that data is striped across all available network links and servers. In order to achieve this, we used:

- multiple clients,
- multiple threads (10-20),
- large block sizes (10MB and more)

With the sufficient degree of parallelism and sufficient block size, we were able to achieve satisfactory performance, nearly matching what is possible to achieve locally: Write operations reached 1GB/s, saturating 10Gbit/s link (Figure 3). Read performance was lower, but still very good, reaching 700MB/s.
The authors consulted BeeGFS support team about write:read disparity. This behavior is often observed, due to the characteristic that results in write operations being cached and reorganised

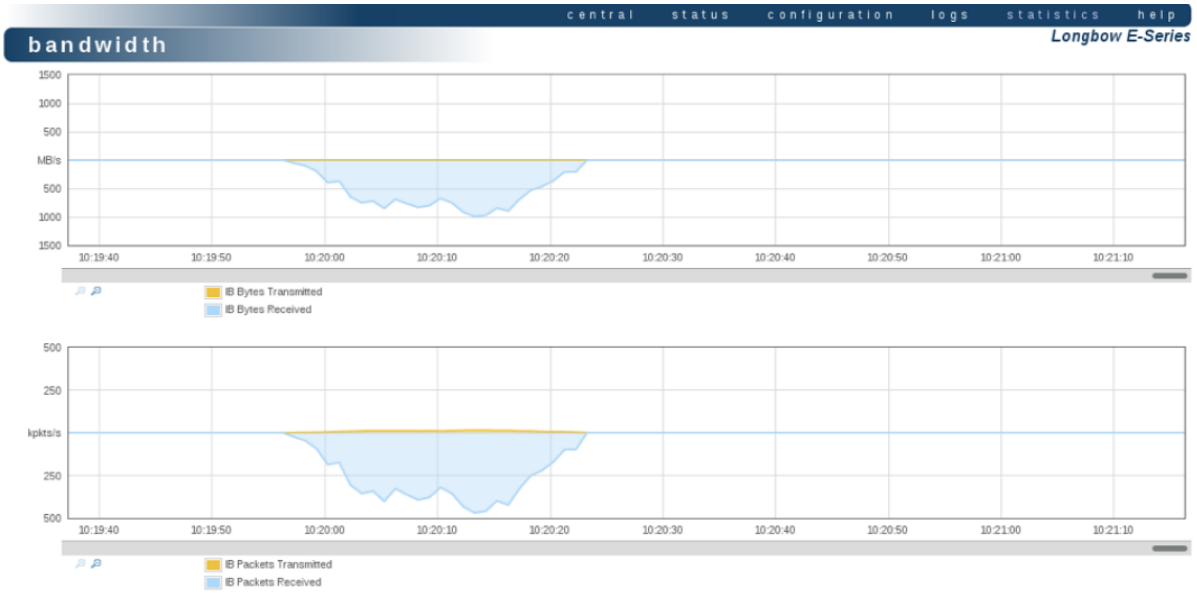


Figure 3. Network utilization diagram for Canberra-Singapore link during BeeGFS data access

more efficiently than read operations. Often, this allows the storage back-end to aggregate write operations in a more efficient manner. When reading data, it is sent out to the clients as quickly as possible, and more data is streamed read from the disks as it is needed. This doesn't allow as much room for optimization as the case of write operations. Such performance profile is not unique to BeeGFS and is often observed with other parallel filesystems, such as Lustre [4] [15].

3.2. Geopipeline

Geopipeline is a set of tools implementing computational genomics codes on a set of geographically distributed hardware. It consists of two main components: ElastiCluster and Biopipeline - and also relies on a parallel file system - in our case BeeGFS

3.2.1. ElastiCluster

ElastiCluster is a software suite implementing on-demand HPC Cluster capability in the Cloud. By default it uses Ansible configuration management, NFS file sharing and Sun Grid Engine job scheduler [3]. All these components can be customized as required.
We configure OpenStack and ElastiCluster in a way which supports running a BeeGFS storage cluster in a central location and then a geographically distributed HPC cluster spanning multiple locations which connect to the central storage cluster.
After the relevant configuration has occured as detailed in Section 2.2 and Section 2.3, the geographical location of the compute instances becomes irrelevant. When ElastiCluster is instantiated, it deploys a head node in the central location and a number of virtual compute nodes spread across continents. The compute nodes then mount the remote BeeGFS parallel file system and report to the Sun Grid Engine scheduler running on the head node and are ready to run jobs. Physical distance between the nodes running virtual instances is abstracted away and the applications run and behave exactly in the same way as if it were running on a local cluster hosted in a single datacentre rack. (Figure 4) and (Figure 5) represent a state of the virtual cluster after launch and during processing.

Figure 4. Ganglia monitoring interface after starting a geographically distributed cluster

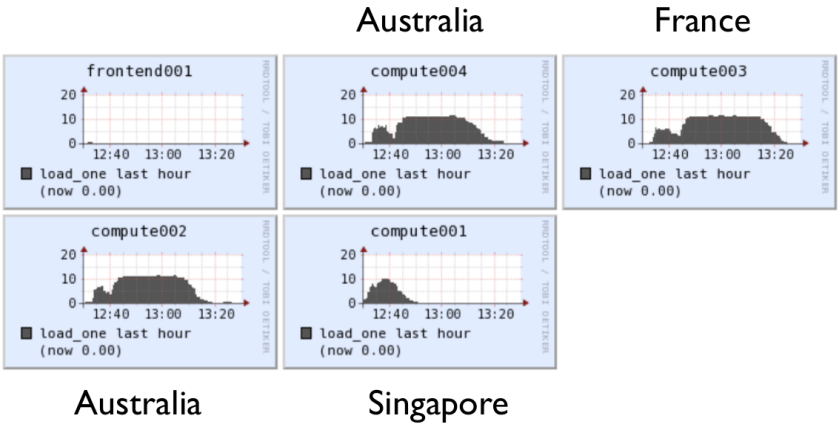


Figure 5. Resource utilization across the distributed cluster members

3.2.2. Implementation of variant calling genome analysis pipeline

Next, we demonstrate the on-the-fly provisioning and setup of a virtual machine which can be used to parallelize a genomic analysis workflow. We selected a clinically relevant workflow, called variant calling, which takes genomic sequences from cancer samples and detects mutations in genes that could be used to determine the prognosis of a patient, or to identify potential chemotherapy drugs that could be used for treatment. Because each cancer sample can be analysed separately, the workflow is amenable to simple asynchronous parallelization without any interprocess communication. Each compute node is configured with 12 CPU cores, 24GB RAM, 20GB system disk and 100GB ephemeral scratch space.

In this workflow, genomic sequences are processed in a pipeline through a series of steps using different applications to identify and annotate mutations (Figure. 9). We use a pipeline appli-

cation to orchestrate the steps in processing and to distribution the processing to the compute nodes using the SGE scheduler.

(Figure. 6) and (Figure.7) illustrate typical local CPU and memory utilization on one node during example run of the above pipeline. These metrics were recorded using Ganglia. Figure 8 captures typical aggregate I/O usage patterns measured against the parallel file system using sysstat package. These graphs clearly show that these applications are primarily CPU bound and in certain steps (particularly alignment) storage intensive, both in terms of read and write. All applications are configured to run a number of threads matching the number of available cores. Memory utlization is moderate. Network utilization is high at times of heavy storage utilization and very low at all other times. Native RDMA is used for data movement. TCP over IPoIB is used for SGE control streams and monitoring - all these components combined use only minimal network bandwidth, measured in kilobytes a second.

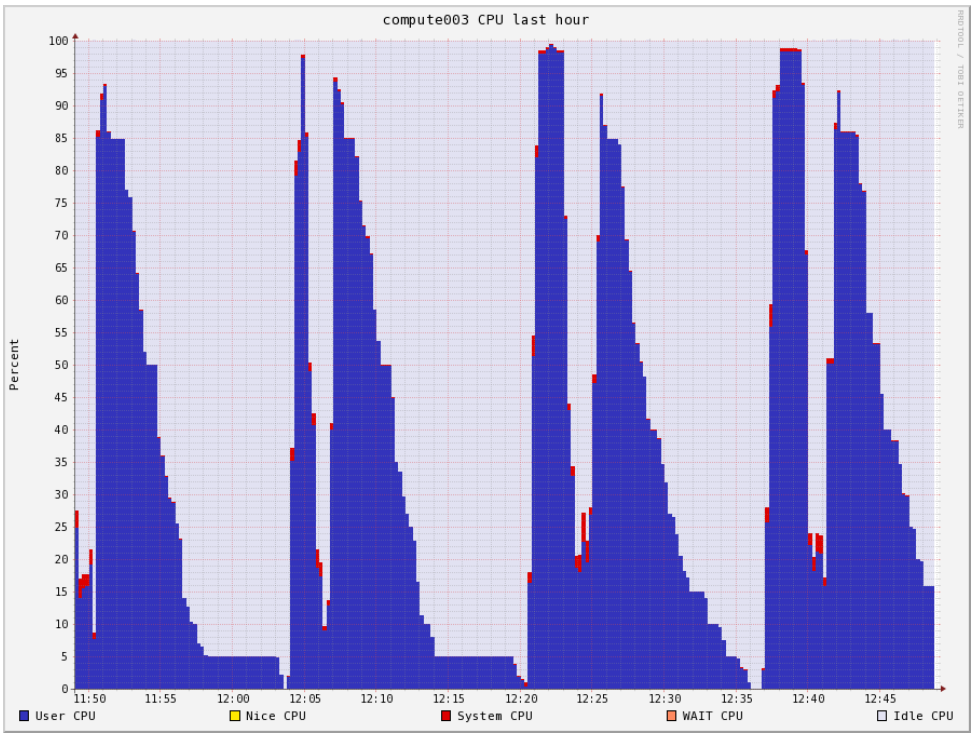


Figure 6. CPU utilization on a node across many test iterations

1. Genomic sequences from each cancer sample are processed with an aligner - an application that compares the sequences to a human reference genome sequence and identifies the position and alignment of each sequence from the cancer samples.
2. The files from each cancer sample are processed by a variant caller program, which compares the aligned sequences to the human reference genome sequence to identify variations (substitutions, insertions, deletions) in the cancer samples.
3. The variant files from each cancer sample are annotated. A specialized application compares each variation to multiple databases to identify what potential effects of each mutation have on regions in the genome.

The applications are pre-installed in the VM images together with their dependencies to enable portability. An example output is shown on (Figure 3.2.2). The reference datasets required by the aligner, variant caller, and annotation tool, are located on remotely mounted BeeGFS. Depending on the expected data access pattern, the datasets can be accessed in place or staged

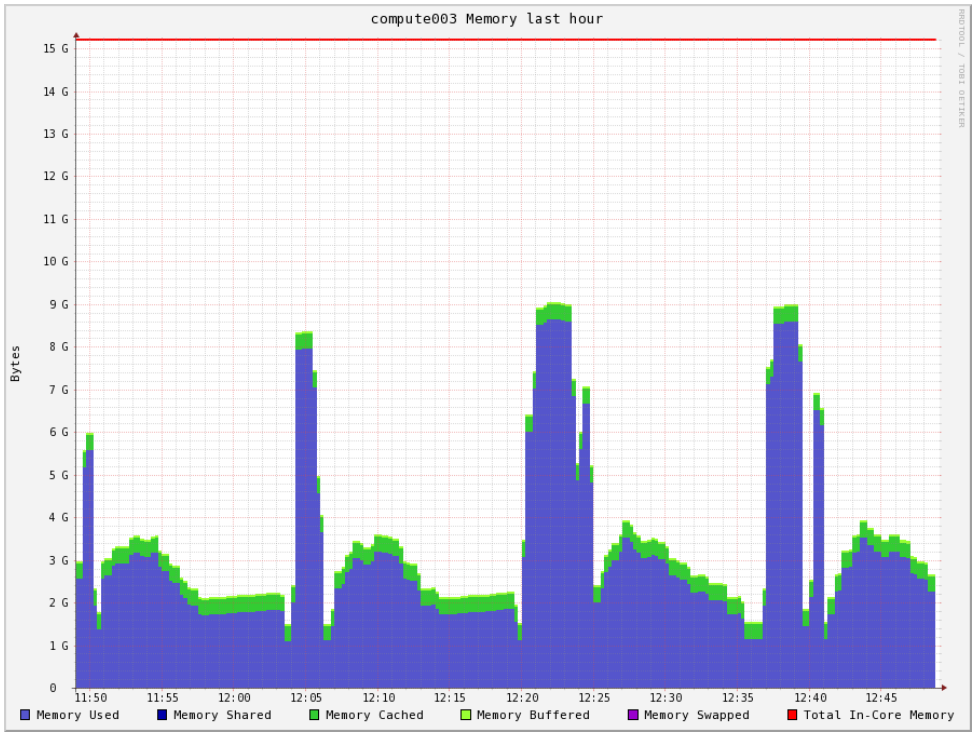


Figure 7. Memory utilization on a node across many test iterations

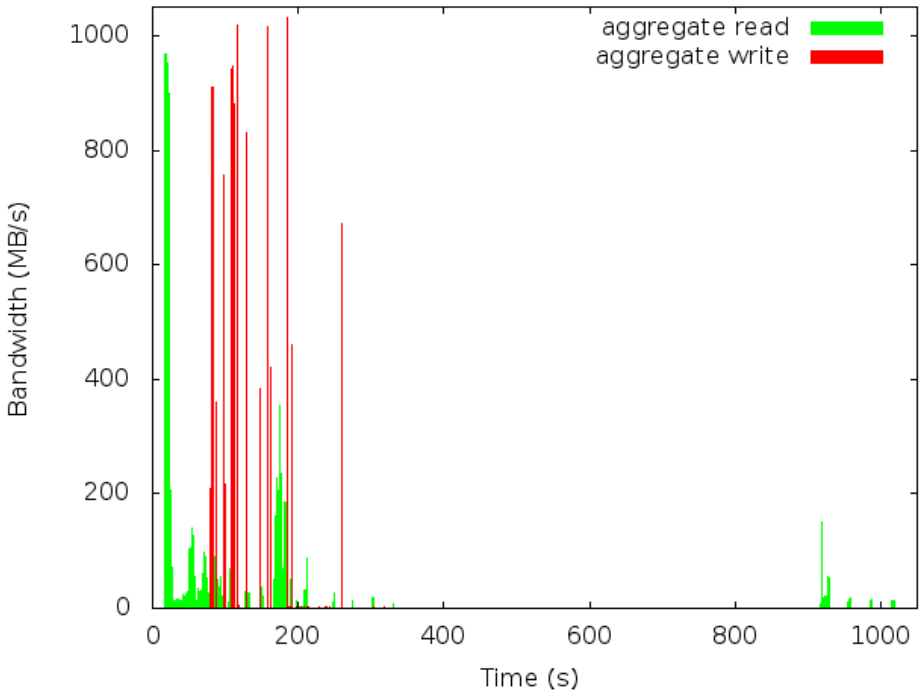


Figure 8. Parallel filesystem read/write bandwidth across single test iteration

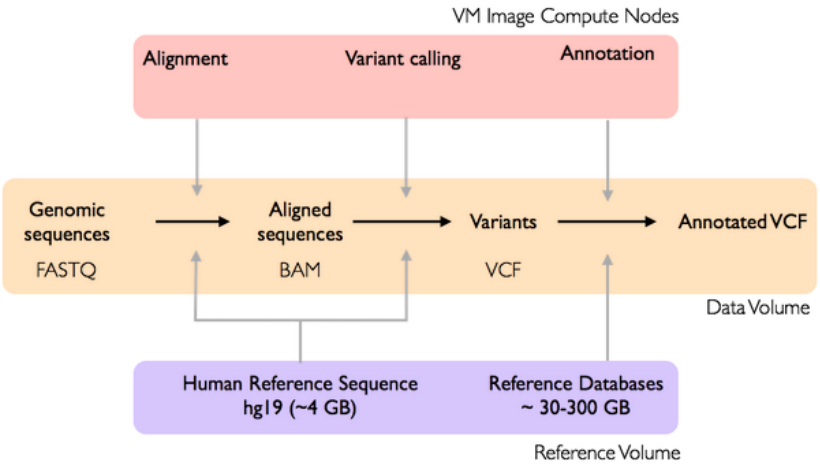


Figure 9. Workflow for variant calling of genomic data from cancer samples

into local scratch. After the processing is complete, the output is also saved on the BeeGFS parallel file system.

```
================================================================================
                    Starting Pipeline at 2016-03-15 21:04
================================================================================

================================= Stage align [AD365_S3_L001]==================
...
============================== Stage variant [AD363_S2_L001] ==================
...
============================ Stage annotate [AD363_S2_L001] ==================
...
======================================== Pipeline Finished ====================
21:48:07 MSG:  Finished at Tue Mar 15 21:48:07 UTC 2016
21:48:07 MSG:  Outputs are:
                annotate/AD407_S5_L001.avinput
                annotate/AD407_S5_L001.hg19_multianno.csv
                annotate/AD363_S2_L001.hg19_multianno.csv
                annotate/AD422_S6_L001.avinput
                annotate/AD422_S6_L001.hg19_multianno.csv
                ... 8 more ...
```

Listing 7. Example output from the pipeline

3.2.3. Geopipeline performance analysis

Performance metrics gathered during the test show that the execution time is determined mostly by CPU performance and, while average I/O bandwidth is low, it can be very streaming read and write intensive in short burts. Given highly parallel nature of the workload, performance can be further optimized by increasing the number of cores available for processing, increasing per-core performance and at later stage increasing storage performance.
The CPU utilization metrics gathered throughout the run show a large variation between completion times across different sites. This is due to heterogenous hardware setup, particularly:

- difference in per-core CPU performance (up to 50% higher per-core performance between Haswell and Sandy Bridge)

- faster local data staging (Singapore nodes don't have to compete for WAN bandwidth with other sites)
- local storage performance (SSD-equipped machines can achieve 50% higher storage bandwidth)

The main focus of this paper is functionality more than performance. The goal of this research is demonstrating the ability to aggregate geographically distant compute resources and enabling the users to efficiently work with large amounts of data over large distances, hence we don't see performance disparity as a problem. For this reasons we haven't performed detailed analysis of performance data or attempts to optimize the pipeline further. This can be a subject of further research. More information on performance analysis and comparison between different types of Public and Private Clouds can be found in [7] and [11].

3.3. Extempore

In contrast to the high bandwidth genome analysis pipeline, we also explored the potential of the InfiniCortex network for interactive latency-sensitive applications using the Extempore "live" interactive HPC programming environment [13]. Extempore allows the HPC programmer to hot-swap running code on-the-fly. It supports seamless integration with C/Fortran (C-ABI compatible) in an interactive "live programming" workflow for Exploratory HPC. For the purpose of this demonstration, we ran four Extempore instances with 8 CPU cores, 64GB RAM and 20GB local SSD. All communications used MPI running over IPoIB. We ran 32 MPI processes in total and the code was mostly CPU and communications intensive, with little memory and storage utilization. The emphasis of this experiment was to test the feasibility of real-time interaction with running computations in a geographically distributed environment like ours, so no detailed resource utilization or execution time measurements were taken.
For the Supercomputing '15 event, we ran an interactive plasma physics simulation (Figure 10) from the SC show floor to demonstrate the possibility of interactive steering and simulation across the globe. We used a particle-in-cell distributed (MPI) plasma physics code, based on codes by Viktor Decyk (UCLA) [9]. All aspects of the simulation could be modified:

- re-definition of constants (time step, electric/magnetic fields) and even subroutines (change boundary conditions), via code updates sent from a laptop on the SC show floor
- summary data was streamed back to the laptop on the show floor, running real-time visualisation and sonification of the computation in progress
- updates of the code from the show floor were immediately reflected in the visualisation in real time, so that visitors to the infinicortex booth could see the simulation being "steered" in real-time

The interactive show floor demo was successful, but not without challenges. The InfiniCortex network was reliable and running the MPI codes over multiple locations (e.g. between Singapore and Australia) required no code changes. However, link latency was a significant challenge for this inherently data-parallel application. For this reason, best results were achieved by dividing the computations into sub-problems that were contained within a single site, and then handling result aggregation in an extra tier.
Thanks to the ability of on-demand, interactive access to high performance computing resources distributed around the globe, InfiniCloud2.0 provided a valuable testing environment for Extem-

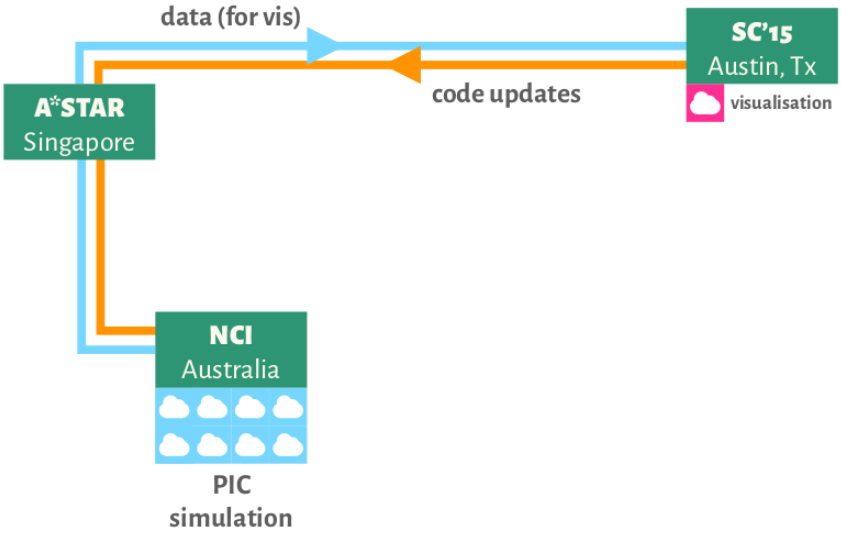


Figure 10. Extempore workflowh

pore, allowing scientific programmers to explore the response of the codes to different parametrisations and workloads.
Future optimization work could involve porting the simulation to reconfigure MPI running over IPoIB to native RDMA which is easier to tune appropriately to the characteristics of a long-distance link. The application could also be enhanced by adding topology and scheduling awareness, which would be very helpful in making sure that each component of the simulation runs in an optimal location, based on latency and bandwidth available. This could then become a foundation for a tiered model for Extempore, where problem sizes are devided into tightly coupled and loosely coupled parts which then can be efficiently scheduled into local and remote worker nodes.

Conclusions

In (Section 1) and (Section 2) we demonstrated the concept, design and implementation of a geographically distributed, High Performance Cloud system, capable of aggregating high performance computing resources available across four continents.
These resources can then be accessed through a uniform set of CLI, GUI and API interfaces, allowing users to create on-demand virtual supercomputers and storage systems, all connected with native InfiniBand.
This clearly demonstrates that it is indeed possible to fully aggregate capacity of a globally distributed pool of computational and storage resources. This is the first, small scale, implementation of a Galaxy of Supercomputers [14].
The InfiniCloud 2.0 platform has proved efficient and resilient. OpenStack components were able to seamlessly communicate over the IPoIB links presented by the InfiniCortex and apart from the image caching overhead, we did not observe any impact of the distance on Cloud operations. Centralized architecture allowed easy scaling to the growing number of InfiniCortex sites, without multiplying the management overhead. Such design enforces consistency through its simplicity - the central site provides a single source of truth.
In this paper, we run four distinct sites and we envision this number can grow to up to ten sites which would allow to provide a good global coverage in data transfer infrastructure for scientific

computations. Hence we are confident to state that our current design scales sufficiently for its purpose.

It is worth noting that this design made the Singapore site a single point of failure. While this is an acceptable risk for a prototype, semi-production environment, a full-production, large scale system would require a slightly different approach providing greater operational resiliency, however providing such solution is beyond the scope of this paper.

In (Section 3.1) we present BeeGFS storage optimized for high-latency, high-bandwidth links proved to be a very powerful tool, supplementing data processing toolkit with the ability to access data over long distance. This means that data collections no longer need to be copied for processing and can be accessed in place.

ElastiCluster and Biopipeline can very easily adapt to operate in a globally-distributed system. After some I/O optimizations, we were able to transparently distribute computational genomics jobs across four continents, aggregating the entire available capacity and creating a fully functional, global HPC cluster, realising the vision of a Galaxy of Supercomputers.

The distributed plasma physics simulation in Extempore demonstrated the ability of the Infinicloud2.0 network to support real-time bidirectional data streaming and real-time code hot-swapping. As dynamic cloud HPC contexts become more popular (e.g. [10] and [16]) the issues of on-demand interactivity and real-time feedback are active areas of research.

We believe a novel approach to High Performance Computing and Cloud Computing proposed in this paper can enable new ways of utilizing computational resources, joining resources available in multiple locations and providing new, more efficient ways of interacting with data collections.

*This work was supported by the A*STAR Computational Resource Centre through the use of its high performance computing facilities.*

This research was undertaken with the assistance of resources from the National Computational Infrastructure (NCI), which is supported by the Australian Government.

The authors wish to thank Universite de Reims Champagne-Ardenne for providing the equipment essential for the experiments described in this paper.

The authors wish to thank Stony Brook University, New York for providing the equipment essential for the experiments described in this paper.

The authors wish to thank Fraunhofer Institute and in particular Bernd Lietzow, Sven Breuner, Frank Kautz and Christian Mohrbacher for their contribution and generous support for our BeeGFS work.

References

1. Amazon EC2 pricing. `http://aws.amazon.com/ec2/pricing/`, urldate = 2016-05-20.

2. BeeGFS parallel filesystem. `http://www.beegfs.com`.

3. ElastiCluster. https://github.com/gc3-uzh-ch/elasticluster.

4. Lustre File System, Operations Manual - Version 2.0. http://wiki.old.lustre.org/manual/LustreManual20_HTML/, urldate = 2016-05-20.

5. OpenStack Cloud Computing Platform. http://www.openstack.org.

6. PuppetForge OpenStack Puppet Modules. https://forge.puppet.com/puppetlabs/openstack.

7. Joseph Antony, Jakub Chrzeszczyk, Dongyang Li, Matthew Sanderson, Andrzej Chrzeszczyk, and Ben Evans. An Initial Microbenchmark Performance Study for Assessing the Suitability of Scientific Workloads Using Virtualized Resources from a Federated Australian Academic Cloud & EC2. Presented at HPC in Asia poster session at International Supercomputing Conference 2014, Leipzig, Germany.

8. Kenneth Ban, Jakub Chrzeszczyk, Andrew Howard, Dongyang Li, and Tin Wee Tan. InfiniCloud: Leveraging Global InfiniCortex Fabric and OpenStack Cloud for Borderless High Performance Computing of Genomic Data and Beyond. *Supercomputing Frontiers and Innovations 2015, Vol2.*

9. V. Decyk. Skeleton Particle-in-Cell Codes on Emerging Computer Architectures. *Computing in Science Engineering*, PP(99):47–52, 2015.

10. Marius Hillenbrand, Viktor Mauch, Jan Stoess, Konrad Miller, and Frank Bellosa. Virtual InfiniBand clusters for HPC clouds. In *Proceedings of the 2nd International Workshop on Cloud*. ACM, 2012.

11. Jonathan Low, Jakub Chrzeszczyk, Andrew Howard, and Andrzej Chrzeszczyk. Performance Assessment of Infiniband HPC Cloud Instances on Intel Haswell and Intel Sandy Bridge Architectures. *Supercomputing Frontiers and Innovations 2015, Vol2.*

12. Gabriel Noaje and Marek Michalewicz. Around The Globe Towards Exascale: InfiniCortex Past and Present. Presented at Supercomputing Frontiers Conference, Singapore, 2016.

13. Ben Swift, Andrew Sorensen, Henry Gardner, Peter Davis, and Viktor K. Decyk. Live Programming in Scientific Simulation. 2(4):4–15.

14. Tin Wee Tan, Dominic S.H. Chien, Yuefan Deng, Seng Lim, Sing-Wu Liou, Jonathan Low, Marek Michalewicz, Gabriel Noaje, Yves Poppe, and Geok Lian Tan. InfiniCortex: A path to reach Exascale concurrent supercomputing across the globe utilising trans-continental InfiniBand and Galaxy of Supercomputers. Supercomputing Frontiers Conference 2015, Singapore.

15. Calleja Paul Turek, Wojciech. Technical bulletin: High performance lustre filesystems using dell powervault md storage. http://i.dell.com/sites/content/business/solutions/hpcc/en/Documents/lustre-hpc-technical\%20bulletin-dell-cambridge-03022011.pdf.

16. Jerome Vienne, Wasi-ur Rahman, Nusrat Sharmin Islam, Hari Subramoni, and D.K. Panda. Performance Analysis and Evaluation of InfiniBand FDR and 40GigE RoCE on HPC and Cloud Computing Systems.

DOI: 10.14529/jsfi160205

Making Large-Scale Systems Observable — Another Inescapable Step Towards Exascale

Dmitry Nikitenko[1], *Sergey Zhumatiy*[1], *Pavel Shvets*[1]

The effective mastering of extremely parallel HPC system is impossible without deep understanding of all internal processes and behavior of the whole diversity of the components: computing processors and nodes, memory usage, interconnect, storage, whole software stack, cooling and so forth in detail. There are numerous visualization tools that provide information on certain components and system as a whole, but most of them have severe issues that limit appliance in real life, thus becoming inacceptable for the future system scales. Predefined monitoring systems and data sources, lack of dynamic on-the-fly reconfiguration, inflexible visualization and screening options are among the most popular issues. The proposed approach to monitoring data processing resolves the majority of known problems, providing a scalable and flexible solution based on any available monitoring systems and other data sources. The approach implementation is successfully used in every-day practice of the largest in Russia supercomputer center of Moscow State University.

Keywords: scalable monitoring visualization, situational screen, supercomputer state visualization, joint monitoring sources, supercomputer dashboard, HPC instrument control board.

Introduction

The effective mastering of large-scale supercomputer systems which includes many aspects of management and administering, is impossible without deep understanding of peculiarities of system behavior on all levels. Most of existing techniques and tools require extensive tuning to fit even present system scales and tasks. Taking steps towards Exascale predetermines the strong need for effective highly-scalable and flexible techniques and algorithms that support simultaneous use of a number of data sources with diverse output formats with a dynamical reconfiguration feature as well as means for visualization of obtained data in user-defined combinations and a variety of screening cases and templates, including implementation for mobile devices. At present numerous monitoring systems and visualization tools are available. Most of them were developed for a certain purpose such as network monitoring, without taking into the account extreme parallelism of observed objects and strict scalability requirements.

Zabbix is a system designed to monitoring network services and applications [1] with comprehensive facilities for visualization of observed object state and history of changes. The developers declare support for 10 000+ observed objects and more, but with a consequent reduced monitoring rate to once per several minutes for every attribute, that is hardly acceptable even for present scales of supercomputers, say nothing of Exascale. Moreover there are issues with simultaneous visualization of multiple characteristics and lack of HPC-specific component support "out of the box", such as resource managers support, queuing systems, etc. Some of the issues cannot be fixed by system Zabbix design.

Nagios is another monitoring system [2] for network services. The system provides a wider support for different modules due to log evaluation history. Basic Nagios has very poor tools for visualization, which is why external tools are used. These tools are usually designed to meet certain requirements, and they are rarely flexibly configurable. Moreover, the declared scalability is even worse than thousands of objects. There is Nagios-based commercial software with mature

[1]Research Computing Center M.V. Lomonosov Moscow State University

visualization facilities and some HPC-relevant modules, but stillit is as limited in scalability as the original version.

Moreover, there are many other monitoring systems and data collectors with different visualization facilities: *Ganglia, Collectd, Cacti, OpenNMS, Munin, Monit, NetXMS*, etc. All of them have demerits in flexibility of visualization configuration and difficulties in introducing new data sources.

Among others one should notice *Open Lorenz* by *Lawrence Livermore National Laboratory*. This tool allows every user designing his own web-page with the information on the latest and forthcoming events, job queue state, overall system load rate and some other information. Every data type is available in separate portlet visual element. As for now, custom configuration of portlet combination and positions is available. The number of available characteristics is rather small and the majority of those available do not suit most other supercomputers, fitting the specific certain HPC center workflow. It is very promising that the project is open source and one can expect further development in functionality extension, introducing new portlets, but there have been not many changes during the last 1,5 years [3, 4].

Another interesting project has been developed by *National Center for Supercomputing Applications, University of Illinois*. It is aimed at visualization data on certain jobs behavior screening characteristics of network usage, CPU utilization, etc. and is oriented to users, lacking valuable information on infrastructure or queuing that is important for system administrators. Unfortunately, the project is not publicly available [5]. As we see, the need for efficient HPC dashboard exists and there are same attempts to develop such tools. At present, most approaches lack flexibility, portability and have poor support for diverse data sources and visualization schemes, not to mention critical scalability issues.

1. Design Principles

In our approach we put emphasis on the development of portable, configurable and scalable algorithms and principles aimed to provide flexible all-round methods of control over supercomputer complex of any scale. The analysis of many-year experience of running and supporting large HPC systems provided us same basic principles that the development has to follow.

1. **It is imperative to permanently keep track on all components that influence efficiency of large-scale system output.** It is totally wrong to control only compute nodes, real life imposes much more complicated set of observed objects:
   - Computing hardware: nodes, CPUs, memory stack, disks and storage hierarchy, networks, etc.
   - Infrastructure hardware: this part of hardware is rarely paid enough attention, but it is much more fault-tolerance critical, than compute hardware. It includes cooling system: chillers, heat exchangers, air conditioners; piping, pumps; a set of components of the power system in conjunction with an uninterruptible power supply; fire safety systems and smoke removal; access control.
   - Whole software stack: OS parameters, package and license usage rates and limitations, etc.
   - Dynamics and resource utilization of all user applications.
   - Job queuing from different points of view: currently run jobs study and analysis of the queued jobs structure including issues of simultaneous jobs interference.

- Users. At one hand, all HPC systems are designed for users, and on the other job queue structure, application peculiarities and as a total an output of supercomputer is determined by user activity.

2. **Support extreme levels of parallelism.** One of the distinctive features of the modern computer world is a rapidly growing degree of parallelism in all elements of architecture. Even today, the number of processor cores in the largest system is more than 3 million. Many other options like networks interfaces are in a close range too. These levels must be supported as a starting point, keeping in mind forthcoming Exascale concurrency levels that are expected to be at least thousands times higher. It is related both to scalable algorithms for data analysis that comes from a huge number of objects, and to the means of visualization.
3. **Minimal induced overhead.** Auxiliary tools must not be integrated deeply into a system and must not be able to influence its functionality. Hence, it is best not to be integrated with monitoring system and data collector agents on compute nodes.
4. **The general and the particular points of view.** At one hand system must provide summary statistics for the whole machines in a space-effective (on a single display) manner, and on the other providing detailed reports on any and every component just in a few clicks, including support of wide range of display types: wide display panels, notebook and desktop displays, tablets and other mobile devices.
5. **Flexible configuration of data sources.** It means easy introduction of new data collector of any output format supporting most popular data types and protocols (http, json, csv, etc.), but not limited to any set of those.

2. Implementation

A variety of monitoring systems and data collectors can serve as data source. The developed system of data acquisition runs over HTTP request via PUT methods. The data is submitted in json or csv format. Thus, the data transmission can be performed even by a simple script with a common curl or wget. Most monitoring systems support exporting data via HTTP in json format. If not, an external program (or perl, ruby, python, etc. script) is used to export data, which allows using HTTP and json. At present, we use information that is acquired from: Collectd monitoring system, epilogue scripts of SLURM resource manager, Octoshell cluster management system [6], Octotron fault-tolerance system [7]. General dataflow is shown in fig. 1.

Incorporation of a new data source is implemented easily, but the one thing we should be aware of is the data size. Large-scale systems already include $\sim 10^4$ of nodes, so saving and processing of the raw data from numerous per-node sensors with high granularity can hang the system. This scalability issue can be resolved with on-the-fly filtering and aggregation. As a rule, real raw data is not too important for visualization and must correspond to the resolution limitations or analysis purposes. After aggregation and filtering, the data size is reduced significantly and can easily be stored for reference and post-processing.

Efficient on-the-fly aggregation and filtration is a challenge, and LapLang (LAPTA [8] Language) tool was developed to tackle it. It's main difference from other similar tools is the ability of dynamical on-the-fly changing of data processing architecture without restarting. To run a LapLang program, a daemon service is started. LapLang program consists of named nodes (hereinafter ll-nodes) for data processing that run simultaneously in low resource conditions). Data

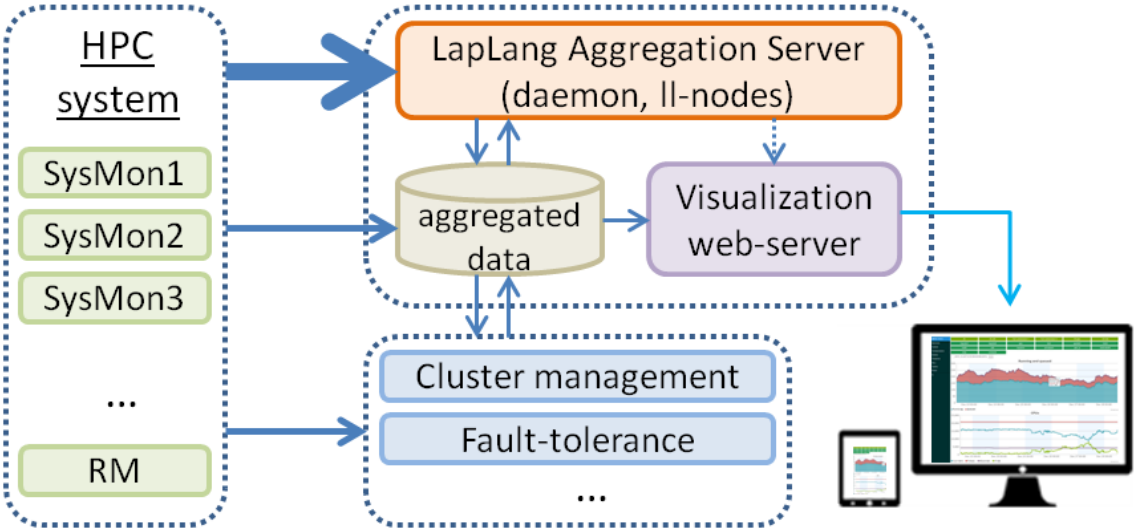


Figure 1. LapLang general dataflow

can be passed from one ll-node to another, generating command pipes. It is allowed to pass data from one ll-node to several, all data flows can be filtered.

Every ll-node processes data portions one by one in FIFO order. Besides the data, ll-node can send and receive commands: to create a new ll-node joining with parent ll-node, to delete specified ll-node with all its links; set filter on the link of two ll-nodes; delete filter from the two ll-nodes link; create link between two specified ll-nodes; delete link between two ll-nodes, get information on all ll-nodes, links and filters (executed only by a head ll-node), finish service (executed by the head ll-node), sent signal "end-of-data" instead of "data".

At the current stage of implementation, the following ll-nodes functionality is available: avg, min, max, file (csv read), http_csv (csv http read), exec (reads program output), outcsv (writes csv to file), slice (sends end-of-data periodically or by condition), agr/grid (aggregation types), sort, join, db_save.

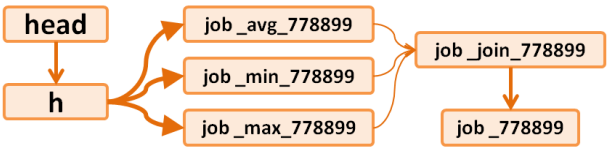


Figure 2. LapLang command pipe aggregating and calculating averages

An example of command pipe that gets min, max and avg for some dynamical characteristic of a job is illustrated in fig. 2. The master "*head*" ll-node performs no processing, "*h*" ll-node obtains data from monitoring system and passes it to three children — *job_avg_778899*, *job_min_778899*, and *job_max_778899*. Only filtered data is passed (bold arrows shown on figure represent filtering by set of ll-nodes). Every filter performs own aggregation by a specified field of data. Later, all three aggregators pass data to *job_join_778899* ll-node, that joins tuples in a new tuple, containing min, max, and avg for a period of time. Next, *job_778899* ll-node saves data to the database. When job finishes, ll-nodes *job_avg_778899*, *job_min_778899*, *job_max_778899*, *job_join_778899* and *job_778899* are killed after processing the last portions of data.

All the data that passed aggregation is saved to database and is immediately available for visualization via web-server. The data is transmitted in a json type and can be easily immediately interpreted by web-client, typically, a web-browser. This allows uniform display methods on

diverse hardware from mobile phones to widescreen panels just with a difference in templates and preferences. Authorization and authentication allows granting access in different scenarios for various user groups and use cases.

Web-page design does not require experience in web-programming and even HTML knowledge. Most portlets are already prearranged, so one just has to add a few lines of slim code into a template. As an example, the following code is used to display a set of available queues as controls, queue load graph and average number of CPU (cores) utilization. The resulting template is shown in fig. 3.

```
== slim :header, locals: {link: '/v1/display/queues'}
  == slim :'components/partitions', locals: {link: '/v1/display/queues'}
  - unless @partition
    == @partition='all'
  div[class="queue" style="height:500px" class="tvz_elem" id="rq"
      tvz_source="queues_rq"
      tvz_display="queues_rq"
      tvz_partition="#{@partition}"
      tvz_title=t("charts.q_rq")
      tvz_labels=t("charts.q_rq_labels")]
  div[class="queue" style="height:500px" class="tvz_elem" id="ctbf"
      tvz_source="queues_ctbf"
      tvz_display="queues_ctbf"
      tvz_partition="#{@partition}"
      tvz_title=t("charts.q_ctbf")
      tvz_labels=t("charts.q_ctbf_labels")]
```

An example of slim-based code to display a set of available queues as controls, queue load graph and average number of CPU (cores) utilization

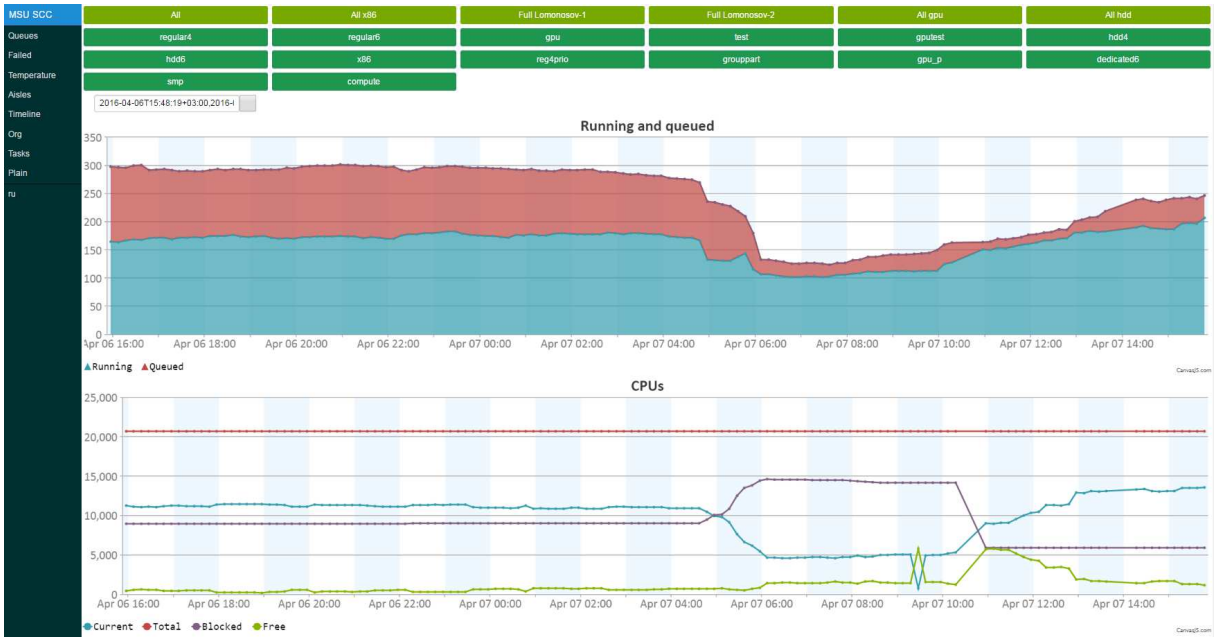


Figure 3. "Lomonosov" system visualization sample: jobs in a queue and utilized core number timeline

The next example illustrates the temprature distribution according to the formal model of the HPC system that includes information on aisles and infrastructure racks location, see fig. 4. Temperature sensors are mapped to the racks. There can be a total of four sensors located on each rack: two upper-mounted and two bottom-mounted. The racks with no sensors are actually racks with compute nodes.

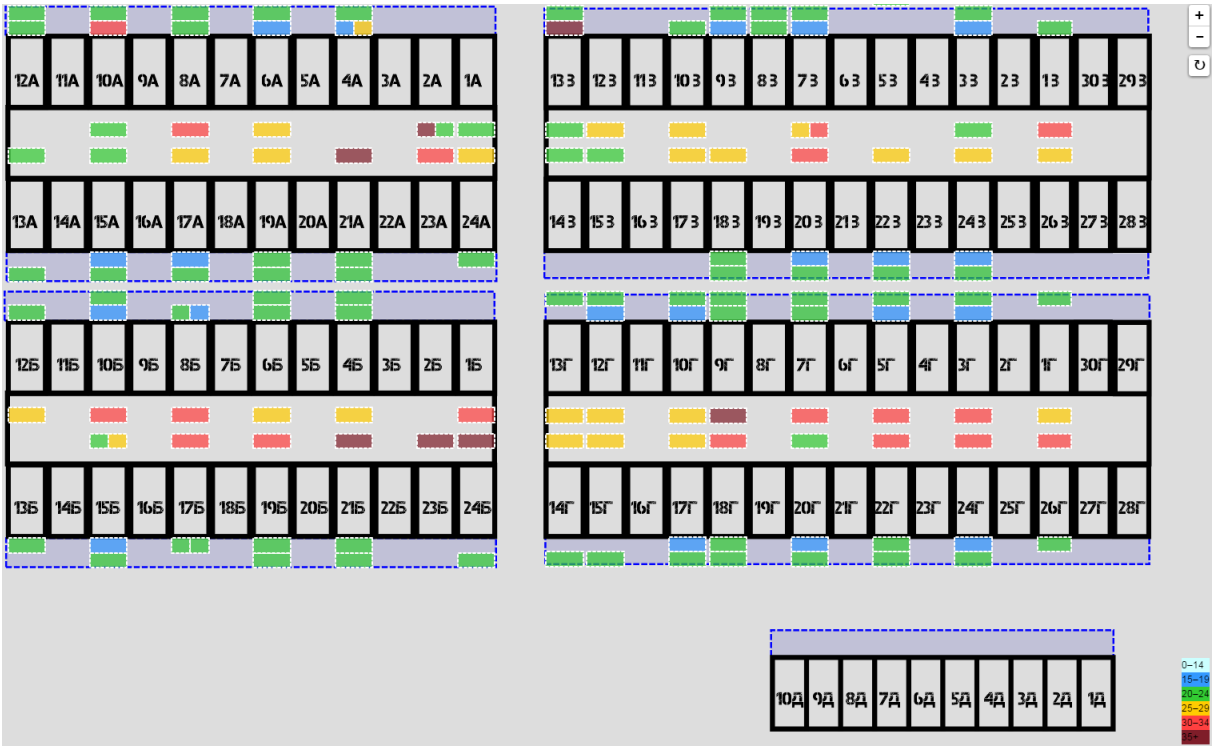


Figure 4. “Lomonosov” system visualization sample: aisles temperatures

The example shown in fig. 5 illustrates the timeline of warning and errors, revealed by OctoTron resilience system.

The developed system also provides data on user activity. As a result of integration with user management system administrator can get in a few clicks user, project or organization details as well as detailed information on any job, including means to reveal categories of user job runs by resource utilization and many other factors.

Conclusions

As a result, the proposed approach allows visualizing diverse data obtained from various data sources on supercomputer functionality, including user activity, hardware state and fault-tolerance notification. The prototype that is being evaluated at Supercomputer Center of Moscow State University can be deployed at any HPC Center with minimal efforts providing a scalable and flexibly configurable tool both for users, and system administrators and managers. The software is developed as open source and will be available for public commits as soon as all the planned features and templates are implemented. The variety of supported data collectors, functionality and visualization templates are definitely to be extended further as the need for such tools on the road to Exascale will only ascend.

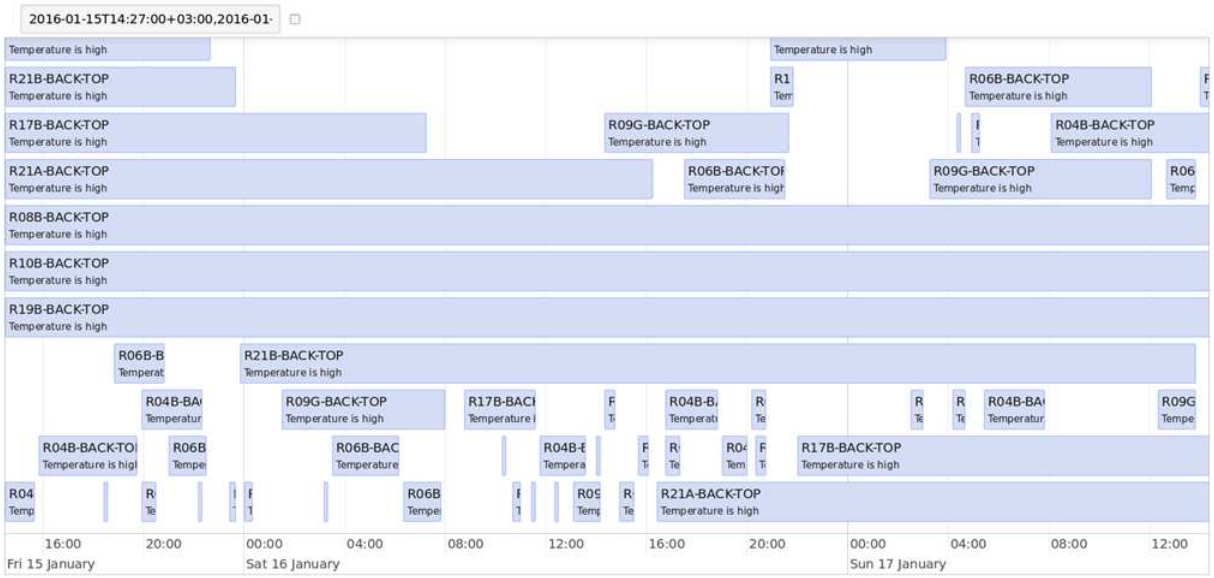


Figure 5. "Lomonosov" system visualization sample: warnings and failures timeline

The work is partially funded by the Russian Foundation for Basic Research, grants 13-07-12206, 2016-07-01199 and by the Ministry of Education and Science of the Russian Federation, Agreement No. 14.607.21.0006 (unique identifier RFMEFI60714X0006).

References

1. Zabbix — The Enterprise-class Monitoring Solution for Everyone, http://www.zabbix.com
2. Nagios — The Industry Standard in IT Infrastructure Monitoring, http://www.nagios.org
3. Long J.W. Lorenz: Using the Web to Make HPC Easier. 2013. 15.
4. OpenLorenz — Web-Based HPC Dashboard and More, https://github.com/hpc/OpenLorenz
5. Showerman M. Real Time Visualization of Monitoring Data for Large Scale HPC Systems // 2015 IEEE International Conference on Cluster Computing. IEEE, 2015. Pp. 706-709.
6. Dmitry Nikitenko, Vladimir Voevodin, and Sergey Zhumatiy. Octoshell: Large Supercomputer Complex Administration System // Russian Supercomputing Days International Conference, Moscow, Russian Federation, 28-29 September, 2015, Proceedings. CEUR Workshop Proceedings, 2015. Vol. 1482. pp. 69-83.
7. Pavel Shvets, Vladimir Voevodin, Sergey Sobolev, Vadim Voevodin, Konstantin Stefanov, Sergey Zhumatiy, Artem Daugel-Dauge, Alexander Antonov and Dmitry Nikitenko. An Approach for Ensuring Reliable Functioning of a Supercomputer Based on a Formal Model. Parallel Processing and Applied Mathematics. 11th International Conference, PPAM 2015,

Krakow, Poland, September 6-9, 2015. Revised Selected Papers, Part I (2016), vol. 9573 of LECTURE NOTES IN COMPUTER SCIENCE, Springer International Publishing, pp. 12-22.

8. Vladimir Voevodin, Anadrey Adinets, Pyotr Bryzgalov, Vadim Voevodin, Sergey Zhumatiy, Dmitry Nikitenko, and Konstantin Stefanov. Job Digest - approach to analysis of application dynamic characteristics on supercomputer systems. Numerical Methods and Programming. 2012. Vol. 13. pp. 160-166.

9. Dmitry Nikitenko, Vladimir Voevodin, Sergey Zhumatiy, Konstantin Stefanov, Alexey Teplov, Pavel Shvets, and Vadim Voevodin. Supercomputer Application Integral Characteristics Analysis for the Whole Queued Job Collection of Large-Scale HPC Systems. Parallel Computational Technologies (PCT'2016): Proceedings of the International Scientific Conference. Chelyabinsk, Publishing of the South Ural State University, 2016. pp. 20-30.

DOI: 10.14529/jsfi160206

Application of CUDA technology to calculation of ground states of few-body nuclei by Feynman's continual integrals method

M.A. Naumenko[1], *V.V. Samarin*[1,2]

The possibility of application of modern parallel computing solutions to speed up the calculations of ground states of few-body nuclei by Feynman's continual integrals method has been investigated. These calculations may sometimes require large computational time, particularly in the case of systems with many degrees of freedom. This paper presents the results of application of general-purpose computing on graphics processing units (GPGPU). The energy and the square modulus of the wave function of the ground states of several few-body nuclei have been calculated using NVIDIA CUDA technology. The results show that the use of GPGPU significantly increases the speed of calculations.

Keywords: NVIDIA CUDA, Feynman's continual integrals method, few-body nuclei.

Introduction

Low-energy reactions involving few-body nuclei [1] constitute a significant part of the studied nuclear reactions. Investigation of their collisions with other nuclei provides valuable information on the mechanisms of fusion and nucleon transfer reactions (*e.g.* [2]). Knowledge of the properties and the ground state wave functions of these nuclei is necessary for the theoretical description of reactions with their participation. The few-body problem in nuclear physics has been studied for a long time. For instance, calculations of ^{3}H and ^{3}He nuclei were performed in [3] based on the Faddeev equations. The expansion in hyperspherical functions (K-harmonics) [4] was used for calculations of ^{3}H nucleus in [5] and ^{4}He nucleus in [6]. In [7] the wave function of the three-body system was obtained using Gaussian basis and the numerical solution of the Hill-Wheeler integral equations.

Feynman's continual integrals method [8, 9] provides a more simple possibility for calculating the energy and the probability density for the ground state of the few-body system, because it does not require expansion of the wave function in a system of functions. This approach may be realized using the Monte-Carlo method with imaginary time and continuous variation of coordinates (*e.g.* [10–12]) or discrete coordinate lattice (*e.g.* within the nuclear lattice effective field theory [13, 14]). The possibility of application of the Monte-Carlo method with imaginary time and continuous variation of coordinates for calculation of energies of ground states of light nuclei up to ^{4}He was declared in [10, 11], but the power of computers available at that time did not allow obtaining reliable results since the statistics was very low. Even today, the authors usually either restrict themselves only to the calculation of energies of ground states of few-body nuclei [12, 13] or perform the more time-consuming calculation of wave functions with large lattice spacing (*e.g.* [14]), which is probably due to the lack of the computing power. In [15] calculations of both energies of ground states and wave functions were performed on the CPU with the statistics 10^5.

In this work an attempt is made to use modern parallel computing solutions to speed up the calculations of ground states of few-body nuclei by Feynman's continual integrals method. The algorithm allowing us to perform calculations directly on GPU was developed and implemented in C++ programming language. The energy and the square modulus of the wave function of the

[1]Joint Institute for Nuclear Research, Dubna, Russian Federation.
[2]Dubna State University, Dubna, Russian Federation.

ground states of several few-body nuclei have been calculated using NVIDIA CUDA technology [16–18]. The results show that the use of GPU is very effective for these calculations.

1. Theory

The energy E_0 and the square modulus of the wave function $|\Psi_0|^2$ of the ground state of a system of few particles may be calculated using continual (path) integrals introduced by Feynman [8, 9]. Feynman's integral

$$K\left(q,t;q_0,0\right)=\int Dq(t)\exp\left\{\frac{i}{\hbar}S\left[q(t')\right]\right\}=\left\langle q\left|\exp\left(-\frac{i}{\hbar}\hat{H}t\right)\right|q_0\right\rangle \tag{1}$$

is a propagator - the probability amplitude for the particle of mass m to travel from the point q_0 to the point q in time t. Here $S[q(t)]$ and $\hat{H}$ are the action and the Hamiltonian of the system, respectively, $Dq(t)$ is the integration measure [8, 9]. For the time-independent potential energy the transition to the imaginary (Euclidean) time $t=-i\tau$ gives the propagator $K_E\left(q,\tau;q_0,0\right)$

$$K_E\left(q,\tau;q_0,0\right)=\int D_Eq(\tau)\exp\left\{-\frac{1}{\hbar}S_E\left[q(\tau')\right]\right\} \tag{2}$$

with the Euclidean action

$$S_E\left[q(\tau')\right]=\int\limits_0^\tau d\tau'\left[\frac{m}{2}\left(\frac{dq}{d\tau'}\right)^2+V(q)\right]. \tag{3}$$

Integration over q with the periodic boundary condition $q=q_0$ allows us to find the energy E_0 of the ground state in the limit $\tau\to\infty$ [10, 11]

$$\int\limits_{-\infty}^{\infty}K_E\left(q,\tau;q,0\right)dq=\mathrm{Sp}\left[\exp\left(-\frac{\hat{H}\tau}{\hbar}\right)\right]=\sum_n\exp\left(-\frac{E_n\tau}{\hbar}\right)+\int\limits_{E_{\mathrm{cont}}}^{\infty}\exp\left(-\frac{E\tau}{\hbar}\right)g(E)dE, \tag{4}$$

$$\int\limits_{-\infty}^{\infty}K_E\left(q,\tau;q,0\right)dq\to\exp\left(-\frac{E_0\tau}{\hbar}\right),\tau\to\infty, \tag{5}$$

$$K_E\left(q,\tau;q,0\right)=\sum_n|\Psi_n(q)|^2\exp\left(-\frac{E_n\tau}{\hbar}\right)+\int\limits_{E_{\mathrm{cont}}}^{\infty}|\Psi_E(q)|^2\exp\left(-\frac{E\tau}{\hbar}\right)g(E)dE. \tag{6}$$

Here $g(E)$ is the density of states with the continuous spectrum $E\geq E_{\mathrm{cont}}$. For the system with a discrete spectrum and finite motion of particles the square modulus of the wave function of the ground state may also be found in the limit $\tau\to\infty$ [10, 11] together with the energy E_0

$$\hbar\ln K_E\left(q,\tau;q,0\right)\to\hbar\ln|\Psi_0(q)|^2-E_0\tau,\tau\to\infty, \tag{7}$$

$$K_E\left(q,\tau;q,0\right)\to|\Psi_0(q)|^2\exp\left(-\frac{E_0\tau}{\hbar}\right),\tau\to\infty. \tag{8}$$

The equation (7) may be used to find the energy E_0 as the slope of the linear part of the curve $\hbar\ln K_E\left(q,\tau;q,0\right)$ calculated for several increasing values of τ. The equation (8) may be used to find the square modulus of the wave function of the ground state $|\Psi_0(q)|^2$ in all points q of

the necessary region by calculating $K_E(q,\tau;q,0)$ at the fixed time τ corresponding to the linear part of the curve $\hbar \ln K_E(q,\tau;q,0)$.

Outside of the classically allowed region the square modulus of the wave function $|\Psi_0(q)|^2$ of the ground state with $E < E_{\text{cont}}$ may be significantly smaller than $|\Psi_E(q)|^2$ for the states with the continuous spectrum $E \geq E_{\text{cont}}$. The ground state term in the formula (6) will not dominate despite the much more rapid decrease of the exponential factors $\exp(-E\tau/\hbar) \ll \exp(-E_0\tau/\hbar), E > E_0$. Therefore, in this case the formulas (7), (8) are in general applicable only for the region not far beyond the classically allowed ground state region.

Such situation may occur in the description of bound states of few-particle systems (*e.g.* two protons and a neutron) when the existence of bound states of some of them (*e.g.* proton plus neutron) is possible.

The contribution of states with the continuum spectrum may be eliminated by introducing infinitely high walls in the potential energy located about the range of the nuclear forces beyond the classically allowed region. Introduction of the boundary condition $\Psi_0(q) = 0$ at these walls will not have a significant effect on the energy E_0 and $|\Psi_0(q)|^2$ far away from the walls.

Feynman's continual integral (2) may be represented as the limit of the multiple integral

$$K(q,\tau;q_0,0) = \lim_{\substack{N\to\infty \\ N\Delta\tau=\tau}} \int\cdots\int \exp\left\{-\frac{1}{\hbar}\sum_{k=1}^{N}\left[\frac{m(q_k-q_{k-1})^2}{2\Delta\tau} - \frac{V(q_k)+V(q_{k-1})}{2}\Delta\tau\right]\right\}\times \\ \times C^N dq_1 dq_2 \ldots dq_{N-1}, \tag{9}$$

where

$$q_k = q(\tau_k), \tau_k = k\Delta\tau, k = \overline{0,N}, q_N = q, C = \left(\frac{m}{2\pi\hbar\Delta\tau}\right)^{1/2}. \tag{10}$$

Here $(N-1)$-fold integral corresponds to averaging over the "path" of the particle as a broken line in the plane (q,τ) with the vertices $(q_k,\tau_k), k = \overline{1,N-1}$.

For the approximate calculation of the continual integral (9) the continuous axis τ is replaced with the grid $\tau = \tau_k = k\Delta\tau, k = \overline{0,N}, N \geq 2$ with the step $\Delta\tau$ and the Euclidean propagator of a free particle $K_E^{(0)}(q,\tau;q_0,0)$ is separated [9, 10]

$$K_E(q,\tau;q_0,0) \approx K_E^{(0)}(q,\tau;q_0,0)\left\langle \exp\left[-\frac{\Delta\tau}{2\hbar}\sum_{k=1}^{N}(V(q_k)+V(q_{k-1}))\right]\right\rangle, \tag{11}$$

$$K_E^{(0)}(q,\tau;q_0,0) = \left(\frac{m}{2\pi\hbar\tau}\right)^{1/2}\exp\left[-\frac{m(q-q_0)^2}{2\hbar\tau}\right]. \tag{12}$$

Requiring $q_N = q_0$, we obtain

$$K_E(q_0,\tau;q_0,0) \approx K_E^{(0)}(q_0,\tau;q_0,0)\left\langle \exp\left[-\frac{\Delta\tau}{\hbar}\sum_{k=1}^{N}V(q_k)\right]\right\rangle, \tag{13}$$

$$K_E^{(0)}(q_0,\tau;q_0,0) = \left(\frac{m}{2\pi\hbar\tau}\right)^{1/2}. \tag{14}$$

Here and below the angle brackets mean averaging of the values of the quantity F

$$F = \exp\left[-\frac{\Delta\tau}{\hbar}\sum_{k=1}^{N}V(q_k)\right] \tag{15}$$

over random trajectories, *i.e.* over $(N-1)$-dimensional vectors $Q = \{q_1, \ldots, q_{N-1}\}$ with the distribution law $W(q_0; q_1, \ldots, q_{N-1}; q_N = q_0)$

$$W(q_0; q_1, \ldots, q_{N-1}; q_N = q_0) = C^{N-1} N^{1/2} \exp\left[-\frac{m}{2\hbar\Delta\tau}\sum_{k=1}^{N}(q_k - q_{k-1})^2\right]. \tag{16}$$

This averaging may be calculated using the Monte Carlo method [19]

$$\langle F \rangle = \frac{1}{n}\sum_{i=1}^{n} F_i, \tag{17}$$

where n is the total number of random trajectories, $n \sim 10^5$ - 10^7.

The standard algorithm for simulation of the random vector consists in a sequential choice of the values of its components from the conditional distributions $W_1(q_1)$, $W_2(q_2|q_1)$, $W_3(q_3|q_1, q_2)$, ..., $W_{N-1}(q_{N-1}|q_1, q_2, \ldots, q_{N-2})$ [20]. Here $W_k(q_k|q_1, q_2, \ldots, q_{k-1})$ is the probability density for the values of the quantity q_k given the values of quantities $q_1, q_2, \ldots, q_{k-1}$. For example, for k=1

$$\begin{aligned} W(q_1) = \int dq_2 \ldots \int dq_{N-1} W(q_0; q_1, q_2, \ldots q_{N-1}; q_N = q_0) = \\ = \frac{1}{\sqrt{2\pi\sigma_1}} \exp\left\{-\frac{1}{2\sigma_1}\left[(\mathrm{M}q_1 - q_1)^2\right]\right\}, \end{aligned} \tag{18}$$

$$\sigma_1 = \frac{\hbar\Delta\tau}{m}\left(1 - \frac{1}{N}\right), \mathrm{M}q_1 = q_0. \tag{19}$$

In the case of k=2

$$\begin{aligned} W_2(q_2|q_1) = \int dq_3 \ldots \int dq_{N-1} W(q_0; q_1, q_2, q_3, \ldots q_{N-1}; q_N = q_0) = \\ = \frac{1}{\sqrt{2\pi\sigma_2}} \exp\left\{-\frac{1}{2\sigma_2}\left[(\mathrm{M}q_2 - q_2)^2\right]\right\} \frac{1}{\sqrt{2\pi\sigma_1}} \exp\left\{-\frac{1}{2\sigma_1}\left[(\mathrm{M}q_1 - q_1)^2\right]\right\}, \end{aligned} \tag{20}$$

$$\sigma_2 = \frac{\hbar\Delta\tau}{m}\left(1 - \frac{1}{N-1}\right), \mathrm{M}q_2 = \left(1 - \frac{1}{N-1}\right) q_1 + \frac{1}{N-1} q_0. \tag{21}$$

Finally, in the general case

$$\begin{aligned} W_k(q_k|q_1, q_2, \ldots, q_{k-1}) = \int dq_{k+1} \ldots \int dq_{N-1} W(q_0; q_1, q_2, q_3, \ldots, q_{N-1}; q_N = q_0) = \\ = \frac{1}{\sqrt{2\pi\sigma_k}} \exp\left\{-\frac{1}{2\sigma_2}\left[(\mathrm{M}q_k - q_k)^2\right]\right\} \ldots \frac{1}{\sqrt{2\pi\sigma_1}} \exp\left\{-\frac{1}{2\sigma_1}\left[(\mathrm{M}q_1 - q_1)^2\right]\right\}, \end{aligned} \tag{22}$$

$$\sigma_k = \frac{\hbar\Delta\tau}{m}\left(1 - \frac{1}{N-k+1}\right), \mathrm{M}q_k = \left(1 - \frac{1}{N-k+1}\right) q_{k-1} + \frac{1}{N-k+1} q_0. \tag{23}$$

Introducing the variable A_k

$$A_k = (N-k+1)^{-1} \tag{24}$$

we obtain that the quantity q_k is normally distributed with the mean value $\mathrm{M}q_k$, variance D_k and standard deviation $\sigma_k = \sqrt{D_k}$ [15]

$$\mathrm{M}q_k = (1 - A_k) q_{k-1} + A_k q_0, \tag{25}$$

$$D_k = (1 - A_k) \hbar\Delta\tau/m, \tag{26}$$

$$\sigma_k = [(1 - A_k)\hbar\Delta\tau/m]^{1/2}. \tag{27}$$

In the simulation the next point q_k of the trajectory is calculated by the formula

$$q_k = \mathrm{M}q_k + \zeta_k\sigma_k, k = \overline{1, N-1}, \tag{28}$$

where ζ_k is a normally distributed random variable with zero mean and unity variance. Sample one-dimensional random trajectories for low $N = 6$ and large $N = 1200$ numbers of time steps are shown in fig. 1a and fig. 1b, respectively.

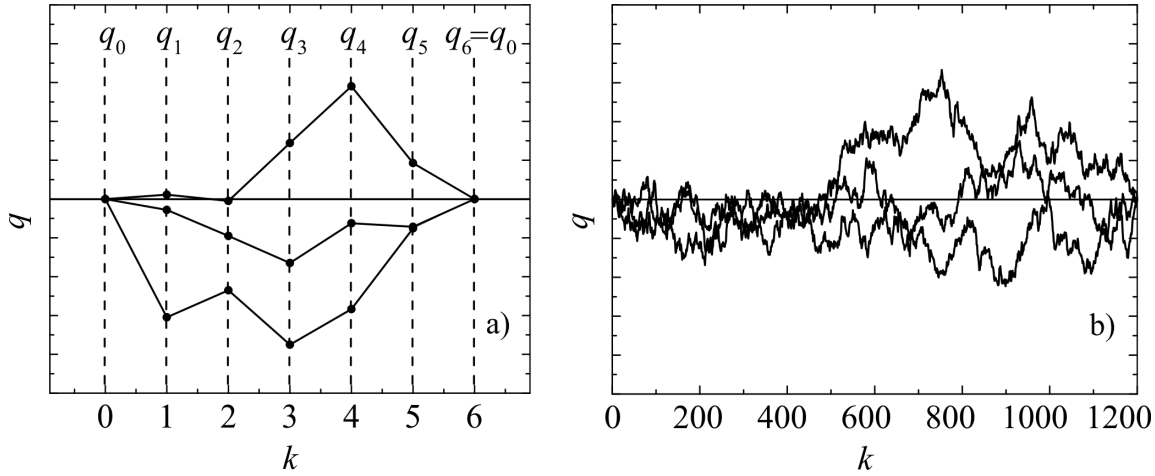


Figure 1. Sample one-dimensional random trajectories for low $N = 6$ (a) and large $N = 1200$ (b) numbers of time steps

For large values of τ random trajectories may reach the region where the probability density for the states with continuum spectrum is substantially larger than the probability density for the ground state, which may lead to a deviation from the asymptotic behavior (7), (8) and the growth of the error. Therefore, the formulas (7), (8) are only applicable for the not very large values of τ.

For convenience of calculations in the scale of nuclear forces we introduce dimensionless variables

$$\tilde{q} = q/x_0, \tilde{\tau} = \tau/t_0, \Delta\tilde{\tau} = \Delta\tau/t_0, \tilde{V} = V(q)/\varepsilon_0, \tilde{E}_0 = E_0/\varepsilon_0, \tilde{m} = m/m_0, \tag{29}$$

where $x_0 = 1$ fm, $\varepsilon_0 = 1$ MeV, m_0 is the neutron mass, $t_0 = m_0 x_0^2/\hbar \approx 1.57 \cdot 10^{-23}$ sec, $b_0 = t_0\varepsilon_0/\hbar \approx 0.02412$. The expressions (7), (8), (13), (16), (25) - (27) may now be represented as

$$\tilde{K}_E(\tilde{q}_0, \tilde{\tau}; \tilde{q}_0, 0) \approx x_0^{-1}\left(\frac{\tilde{m}}{2\pi\tilde{\tau}}\right)^{1/2}\left\langle \exp\left[-b_0\Delta\tilde{\tau}\sum_{k=1}^{N}\tilde{V}(\tilde{q}_k)\right]\right\rangle, \tag{30}$$

$$\tilde{D}_k = \tilde{\sigma}_k^2, \tilde{q}_k = \mathrm{M}\tilde{q}_k + \zeta_k\tilde{\sigma}_k, \tilde{\sigma}_k = x_0[(1 - A_k)\Delta\tilde{\tau}/\tilde{m}]^{1/2}, \tag{31}$$

$$W(\tilde{q}_0; \tilde{q}_1, \ldots, \tilde{q}_{N-1}; \tilde{q}_N) = C^{N-1}N^{1/2}\exp\left[-\frac{1}{2\Delta\tilde{\tau}}\sum_{k=1}^{N}(\tilde{q}_k - \tilde{q}_{k-1})^2\right], \tag{32}$$

$$\frac{1}{b_0}\ln\tilde{K}_E(\tilde{q}, \tilde{\tau}; \tilde{q}, 0) \to \frac{1}{b_0}\ln|\Psi_0(\tilde{q})|^2 - \tilde{E}_0\tilde{\tau}, \tilde{\tau} \to \infty, \tag{33}$$

$$\tilde{K}_E(\tilde{q}, \tilde{\tau}; \tilde{q}, 0) \to |\Psi_0(\tilde{q})|^2\exp\left(-b_0\tilde{E}_0\tilde{\tau}\right), \tilde{\tau} \to \infty. \tag{34}$$

The above formulas are naturally generalized to a larger number of degrees of freedom and few particles including identical ones. The nuclei ^{3}H, ^{3}He and ^{4}He contain no more than two identical fermions (protons and/or neutrons with opposite spins), which ensures that the Pauli Exclusion Principle is satisfied for their ground states. The nucleon identity requires symmetrization of trajectories [11], which is achieved by choosing the Jacobi coordinates in such a way that vectors connect two identical fermions (see below).

It should be noted that the calculation of multiple integrals required to find the multidimensional probability density $|\Psi_0|^2$ by Feynman's continual integrals method continues to be a challenging task. However, the analysis of the properties of $|\Psi_0|^2$ allows us to choose analytical approximations of $|\Psi_0|^2$, *e.g.* as the product of the Gaussian type exponentials. The obtained approximations may be used in dynamic calculations. The application of the formula (7) in a single point in the multidimensional space allows us to find the approximate value of the energy of the ground state.

To reduce the number of degrees of freedom and multiplicity of integrals in the formula (11) the calculation should be performed in the center of mass system using the Jacobi coordinates [4, 9].

For a system of two particles (^{2}H nucleus)

$$\vec{R} = \vec{r}_2 - \vec{r}_1, \tag{35}$$

where $\vec{r}_1$ and $\vec{r}_2$ are the radius vectors of a proton and a neutron, respectively.

For a system of three particles, two of which are identical (2 neutrons or 2 protons in ^{3}H and ^{3}He nuclei, respectively)

$$\vec{R} = \vec{r}_2 - \vec{r}_1, \vec{r} = \vec{r}_3 - \frac{1}{2}\left(\vec{r}_1 + \vec{r}_2\right). \tag{36}$$

In the case of ^{3}H nucleus $\vec{r}_3$ is the radius vector of a proton, $\vec{r}_1$ and $\vec{r}_2$ are the radius vectors of neutrons. In the case of ^{3}He nucleus $\vec{r}_3$ is the radius vector of a neutron, $\vec{r}_1$ and $\vec{r}_2$ are the radius vectors of protons.

For a system of four particles consisting of two pairs of identical particles (2 protons and 2 neutrons in ^{4}He nucleus)

$$\vec{R}_1 = \vec{r}_2 - \vec{r}_1, \vec{R}_2 = \vec{r}_4 - \vec{r}_3, \vec{r} = \frac{1}{2}\left(\vec{r}_3 + \vec{r}_4\right) - \frac{1}{2}\left(\vec{r}_1 + \vec{r}_2\right), \tag{37}$$

where $\vec{r}_1$ and $\vec{r}_2$ are the radius vectors of protons, $\vec{r}_3$ and $\vec{r}_4$ are the radius vectors of neutrons.

The energy of the ground states of bound nuclei is negative $E_0 < 0$, whereas the binding energy E_b (the energy required to disassemble a nucleus into separate nucleons) is positive, $E_b = -E_0 > 0$.

In the calculation of the propagator $K\left(q, \tau; q_0, 0\right)$ for the nuclei ^{2}H, ^{3}H, ^{3}He, ^{4}He neutron-proton $V_{n-p}(r)$, neutron-neutron $V_{n-n}(r)$ and proton-proton $V_{p-p}(r)$ two-body strong interaction potentials have been used. The dependence of the nucleon-nucleon strong interaction with a repulsive core on the distance r was approximated by a combination of Gaussian type exponentials similar to the M3Y potential [21, 22]

$$V_{n-n}(r) \equiv V_{p-p}(r) = \sum_{k=1}^{3} u_k \exp\left(-r^2/b_k^2\right), \tag{38}$$

$$V_{n-p}(r) = \eta V_{n-n}(r). \tag{39}$$

The total interaction potential $V(r) \equiv V_{n-n}(r)$ for two neutrons, $V(r) \equiv V_{n-p}(r)$ for a neutron and a proton, $V(r) \equiv V_{p-p}(r) + e^2/r$ for two protons (here the last term represents the Coulomb part of the potential). The values of the parameters $u_1 = 500$ MeV, $u_2 = -102$ MeV, $u_3 = 2$ MeV, $b_1 = 0.606$ fm, $b_2 = 1.437$ fm, $b_3 = 3.03$ fm and $\eta = 1.2$ provide the absence of bound states of two identical nucleons as well as the approximate equality of the energy $E_b = -E_0$ found from the formula (33) to the experimental values of the binding energies for the nuclei ^{2}H, ^{3}H, ^{3}He, ^{4}He taken from the knowledge base [23] (the comparison is given in tab. 1 below). The plots of the total interaction potential $V(r)$ for two neutrons, a neutron and a proton, and two protons are shown in fig. 2.

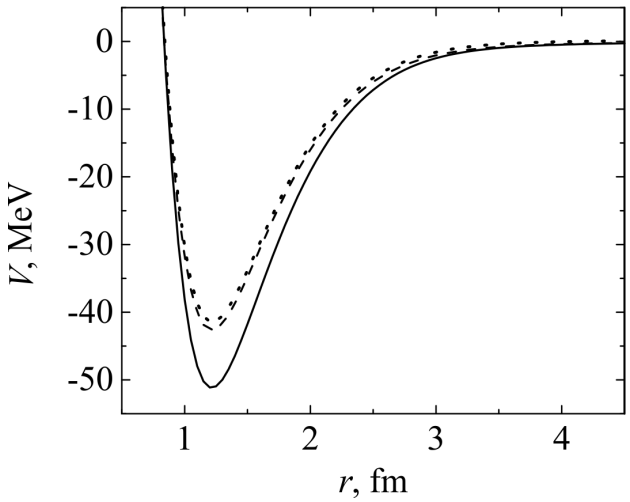


Figure 2. The neutron-proton (solid line), neutron-neutron (dashed line), and proton-proton (dotted line) total interaction potentials $V(r)$

2. Implementation

The Monte Carlo algorithm for numerical calculations was developed and implemented in C++ programming language using NVIDIA CUDA technology. The code samples are not included in the publication, because the algorithm is described in detail in mathematical, physical and implementation aspects, in contrast to *e.g.* [10, 11]. The paper itself is to a great extent the description of the integration method which does not require the use of any additional integration libraries. The detailed description of the algorithm provided allows anyone to easily implement it.

The calculation included 3 steps:

1. $\tilde{K}_E(\tilde{q}, \tilde{\tau}; \tilde{q}, 0)$ was calculated in a set of multidimensional points $\tilde{q}$ (*e.g.* $\left\{\vec{R}; \vec{r}\right\}$ for ^{3}H and ^{3}He nuclei) and the maximum of $\tilde{K}_E(\tilde{q}, \tilde{\tau}; \tilde{q}, 0)$ (*i.e.* $|\Psi_0|^2$) was found.
2. The point $\tilde{q}_0$ corresponding to the obtained maximum was fixed, $\tilde{K}_E(\tilde{q}_0, \tilde{\tau}; \tilde{q}_0, 0)$ was calculated for several increasing values of $\tilde{\tau}$ and the linear region of $\ln \tilde{K}_E(\tilde{q}_0, \tilde{\tau}; \tilde{q}_0, 0)$ was found for calculation of the energy $\tilde{E}_0$ using formula (33).
3. The time $\tilde{\tau}_{\text{lin}}$ corresponding to the beginning of the obtained linear region was fixed and $\tilde{K}_E(\tilde{q}_0, \tilde{\tau}_{\text{lin}}; \tilde{q}_0, 0)$ (*i.e.* $|\Psi_0|^2$) was calculated in all points of the necessary region using formula (34).

The calculation of $\tilde{K}_E(\tilde{q}, \tilde{\tau}; \tilde{q}, 0)$ for the fixed $\tilde{\tau}$ was performed by parallel calculation of exponentials F

$$F = \exp\left[-b_0 \Delta\tilde{\tau} \sum_{k=1}^{N} \tilde{V}(\tilde{q}_k)\right] \tag{40}$$

for every trajectory in a given kernel launch, where $N = \tilde{\tau}/\Delta\tilde{\tau}$.

The principal scheme of the calculation of the ground state energy is shown in fig. 3. The calculation of the propagator (30) is performed using L sequential launches of the kernel. Each kernel launch simulates n random trajectories in the space evolving from the Euclidean time $\tilde{\tau} = 0$ to $\tilde{\tau}_j$, where $j = \overline{1, L}$ (see fig. 1). All trajectories with $N_j = \tilde{\tau}_j/\Delta\tilde{\tau}$ time steps start at the same point $q^{(0)}$ in the space and in the moment $\tilde{\tau}_j$ return back to the same point $q^{(0)}$ according to the probability distribution described above.

The choice of the initial point $q^{(0)}$ is arbitrary for $\tilde{\tau} \to \infty$, but it is clear that for the finite values of $\tilde{\tau}$ available in calculations the point $q^{(0)}$ must be located within the region Ω the integral over which of the square modulus of the normalized ground state wave function is close enough to unity

$$\int_{\Omega} |\Psi_0(q)|^2 dq \approx 1 \tag{41}$$

in order to ensure less number of time steps in the calculation and obtain more accurate results.

All threads in a given kernel launch finish at approximately the same time, which makes the scheme quite effective in spite of the possible delays associated with the kernel launch overhead. Besides, the typical number of kernel launches L required for the calculation of the ground state energy usually does not exceed 100.

Starting from the certain time $\tilde{\tau}_{\text{lin}}$ the obtained values of the logarithm of the propagator $b_0^{-1} \ln \tilde{K}_E$ (30) tend to lie on the straight line, the slope of which gives the value of the ground state energy. The time $\tilde{\tau}_{\text{lin}}$ is then used in the calculation of the square modulus of the wave function.

The principal scheme of the calculation of the square modulus of the wave function is shown in fig. 4. Similarly, the calculation is performed using M sequential launches of the kernel. Each kernel launch simulates n random trajectories in the space evolving from the Euclidean time $\tilde{\tau} = 0$ to the time $\tilde{\tau}_{\text{lin}}$ determined in the calculation of the ground state energy. All trajectories start at the same point $q^{(s)}$ in the space and in the moment $\tilde{\tau}_{\text{lin}}$ return back to the same point $q^{(s)}$ according to the probability distribution described above. Here $s = \overline{1, M}$, where M is the total number of points in the space in which the square modulus of the wave function must be calculated.

One of the benefits of the approach is that the calculation may be easily resumed at a later time. For example, initially the square modulus of the wave function may be calculated with a large space step to obtain the general features of the probability distribution, and later new intermediate points are calculated and combined with those calculated previously. This may be very useful because the calculation of the square modulus of the wave function is generally much more time-consuming since it requires calculation in many points in the multidimensional space.

An important feature of the algorithm allowing effective use of graphic processors is low consumption of memory during the calculation because it is not necessary to prepare a grid of values and store it in the memory.

To obtain normally distributed random numbers the cuRAND random number generator was used. According to the recommendations of the cuRAND developers each experiment was

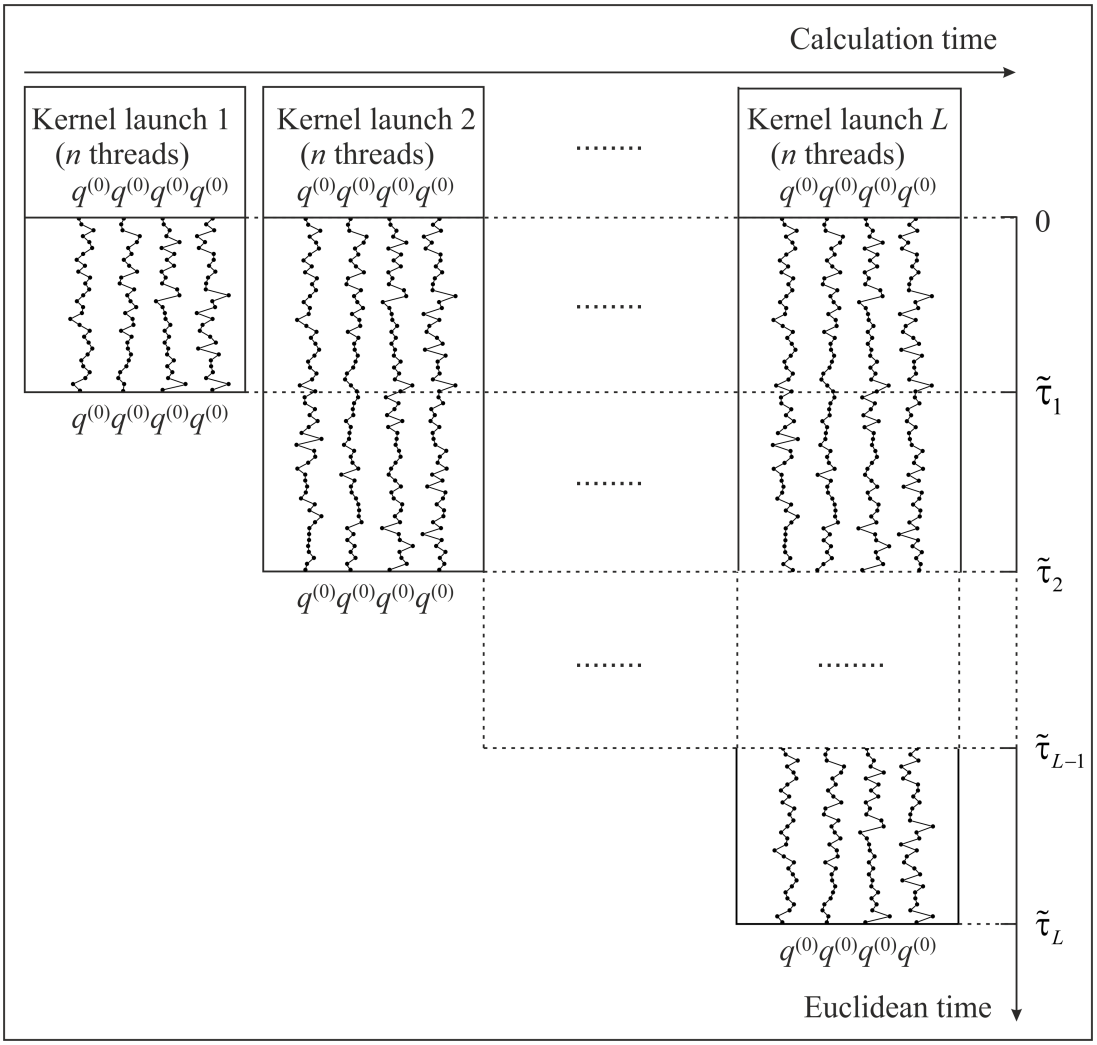


Figure 3. The scheme of calculation of the ground state energy E_0 using formula (33)

assigned a unique seed. Within the experiment, each thread of computation was assigned a unique sequence number. All threads between kernel launches were given the same seed, and the sequence numbers were assigned in a monotonically increasing way.

3. Results and discussion

Calculations were performed on the NVIDIA Tesla K40 accelerator installed within the heterogeneous cluster [24] of the Laboratory of Information Technologies, Joint Institute for Nuclear Research, Dubna. The code was compiled with NVIDIA CUDA version 7.5 for architecture version 3.5. Calculations were performed with single precision. The Euclidean time step $\Delta\tilde{\tau} = 0.01$ was used. Additionally, NVIDIA GeForce 9800 GT accelerator was used for debugging and testing purposes.

The dependence of logarithm of the propagator $b_0^{-1} \ln \tilde{K}_E$ on the Euclidean time $\tilde{\tau}$ is shown in fig. 5 for nuclei ^{2}H (a), ^{3}H (b), ^{3}He (c) and ^{4}He (d). Different symbols correspond to different statistics n: empty circles (10^5), filled circles (10^6, $5{\cdot}10^6$, 10^7).

The behavior of the curves may be easily explained if we note that in all these cases only the energy of the ground state is negative and therefore only the first term in (6) increases with

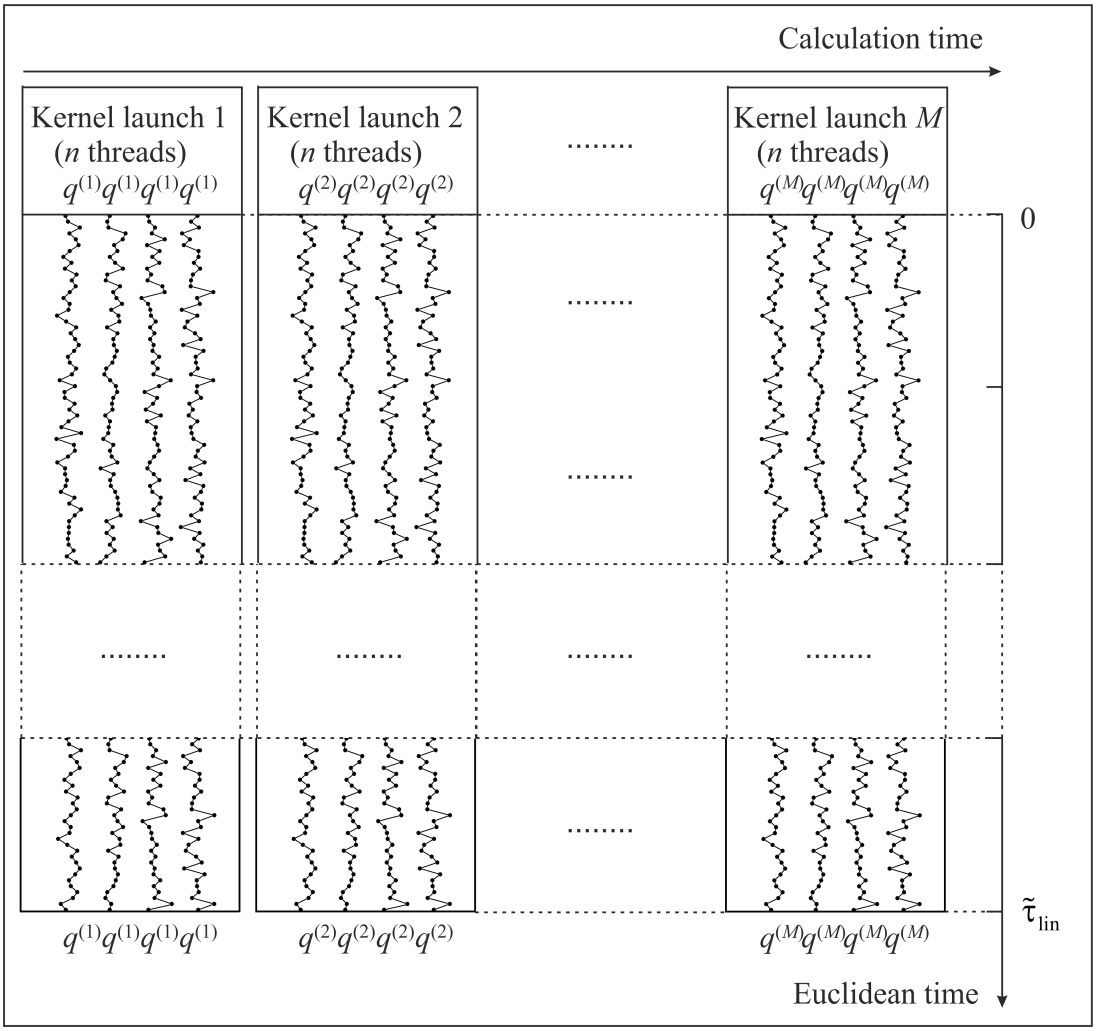


Figure 4. The scheme of calculation of the square modulus of the wave function $|\Psi_0(q)|^2$ using formula (34)

the increase of $\tilde{\tau}$, whereas the energies of the excited states are positive and hence the other terms in (6) decrease with the increase of $\tilde{\tau}$.

The results of linear fitting of the straight parts of the curves are shown in fig. 5e-h. According to the formula (33) the slope of the linear regression equals the energy of the ground state E_0. The obtained theoretical binding energies $E_b = -E_0$ are listed in tab. 1 together with the experimental values taken from the knowledge base [23]. It is clear that the theoretical values are close enough to the experimental ones, though obtaining good agreement was not the goal. The observed difference between the calculated binding energies of ^{3}H and ^{3}He is also in agreement with the experimental values.

The comparison of the square modulus of the wave function for ^{2}H calculated on GPU using NVIDIA CUDA technology within Feynman's continual integrals method and the square modulus of the wave function calculated on CPU within the shell model is shown in fig. 6a. The same potentials (38), (39) were used. Good agreement between the curves confirms that the code based on Feynman's continual integrals method using NVIDIA CUDA technology provides correct results.

It should be mentioned that the wave function cannot be measured directly, though the charge radii and charge distributions obtained from experiments may provide some information

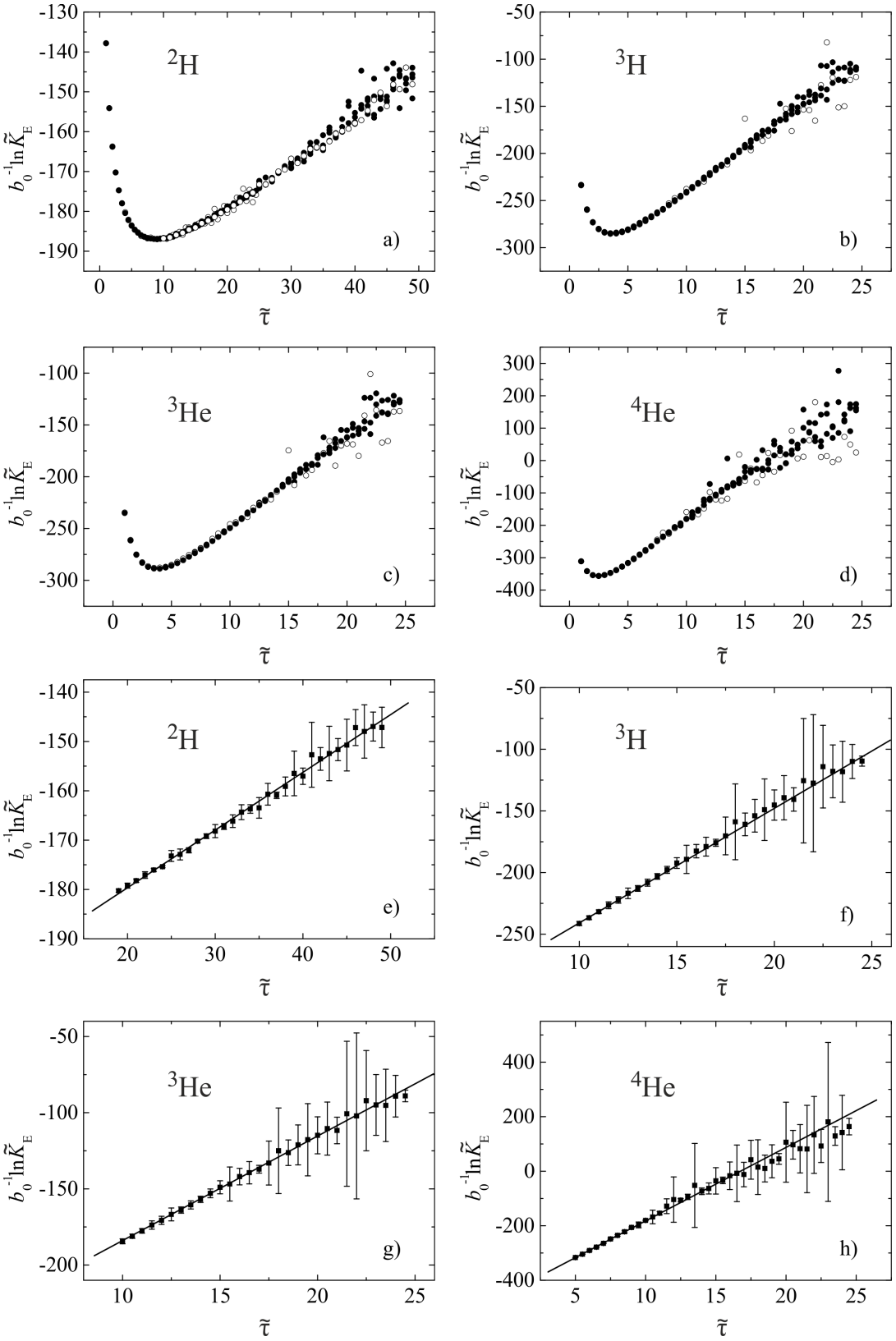


Figure 5. The dependence of the logarithm of the propagator $b_0^{-1}\ln\tilde{K}_E$ on the Euclidean time $\tilde{\tau}$ for ^{2}H (a), ^{3}H (b), ^{3}He (c) and ^{4}He (d). Lines are the results of linear fitting of the data lying on the straight parts of the curves for ^{2}H (e), ^{3}H (f), ^{3}He (g) and ^{4}He (h). Different symbols correspond to different statistics n: empty circles (10^5), filled circles (10^6, $5{\cdot}10^6$, 10^7)

Table 1. Comparison of theoretical and experimental binding energies for the ground states of the studied nuclei

Atomic nucleus	Theoretical value, MeV	Experimental value, MeV
^{2}H	1.17 ± 1	2.225
^{3}H	9.29 ± 1	8.482
^{3}He	6.86 ± 1	7.718
^{4}He	26.95 ± 1	28.296

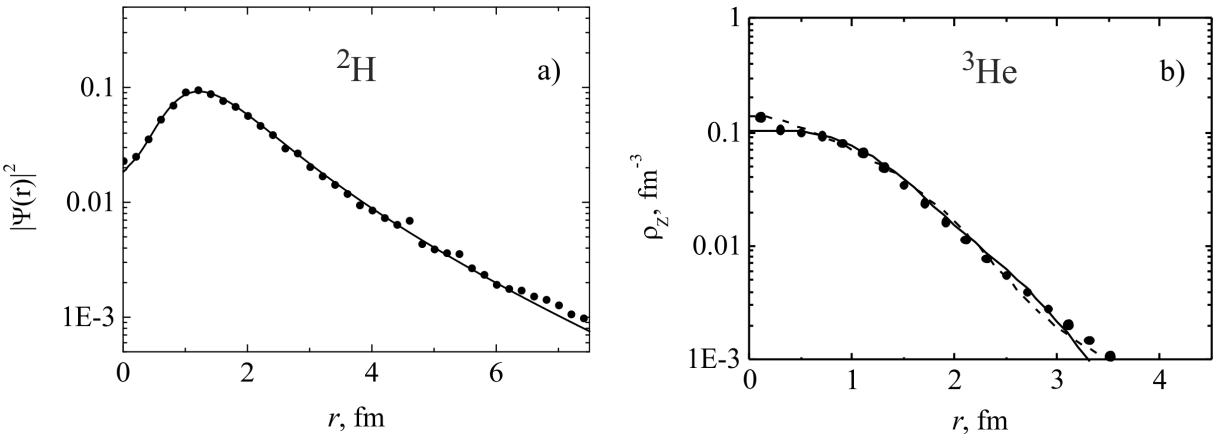


Figure 6. (a) The square modulus of the wave function for ^{2}H calculated on GPU using NVIDIA CUDA technology within Feynman's continual integrals method (circles) compared with the square modulus of the wave function calculated on CPU within the shell model (line); r is the distance between the proton and the neutron. (b) The theoretical charge distribution for ^{3}He (circles) compared with experimental data taken from taken from the knowledge base [23] (lines)

on its behavior. To compare the results of calculations with the experimental charge radii and charge distributions the wave function must be integrated.

The probability density distribution $\left|\Psi_0\left(\vec{R};\vec{r}\right)\right|^2$ for the configurations of ^{3}He nucleus ($p+p+n$) with the angle $\theta = 0°$, $45°$, $90°$ between the vectors $\vec{R}$ and $\vec{r}$ is shown in logarithmic scale in fig. 7a,b,c, respectively, together with the potential energy surface (linear scale, lines). The vectors in the Jacobi coordinates are shown in fig. 7d.

The theoretical charge distribution for ^{3}He obtained by integration of the wave function is compared with experimental data taken from taken from the knowledge base [23] in fig. 6b. As can be seen, the agreement is very good. The obtained theoretical charge radius $\left\langle R_{ch}^2\right\rangle^{1/2} = 1.94$ fm is also very close to the experimental value 1.9664 ± 0.0023 fm.

The probability density distribution for the symmetric tetrahedral configuration of four nucleons in the nucleus ^{4}He

$$\left|\Psi_0\left(\vec{R}_1;\vec{r};\vec{R}_2\right)\right|^2 = \left|\Psi_0\left(R_{1x},0,0;0,0,r_z;0,R_{2y}=R_{1x},0\right)\right|^2, \tag{42}$$

$$\vec{R}_1 \perp \vec{r} \perp \vec{R}_2,\ \left|\vec{R}_1\right| = \left|\vec{R}_2\right|, \vec{R}_1 = \left(R_{1x},0,0\right), \vec{r} = \left(0,0,r_z\right), \vec{R}_2 = \left(0,R_{2y}=R_{1x},0\right) \tag{43}$$

is shown in logarithmic scale in fig. 7e together with the potential energy surface (linear scale, lines). The vectors in the Jacobi coordinates are shown in fig. 7f.

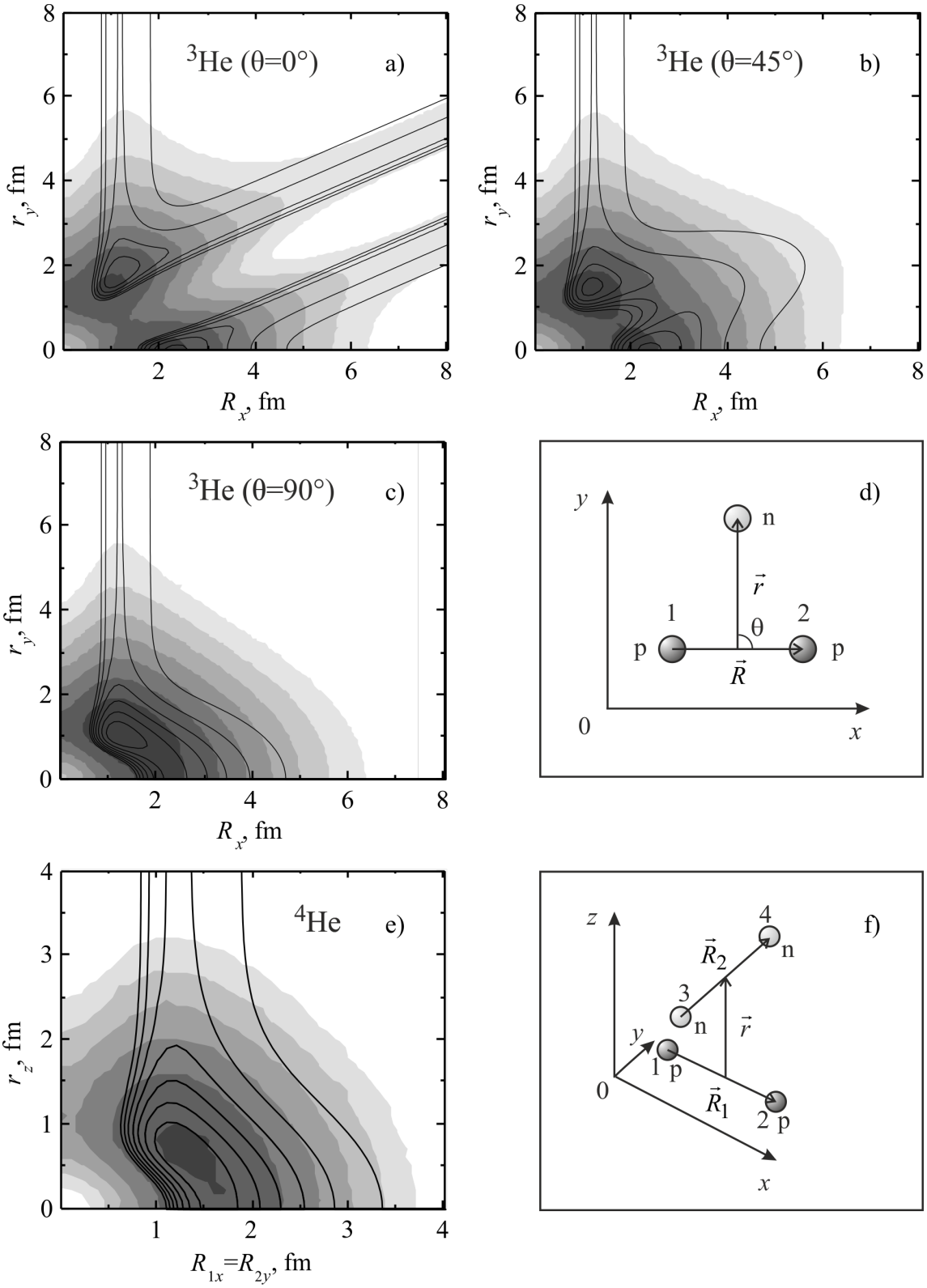


Figure 7. The probability density for the configurations of ^{3}He with $\theta = 0°$ (a), 45° (b), 90° (c) and the vectors in the Jacobi coordinates (d). The probability density for the configuration of ^{4}He symmetric with respect to the positions of protons and neutrons (e) and the vectors in the Jacobi coordinates (f)

Note also that the presence of the repulsive core in the nucleon-nucleon interaction reduces the probability of finding nucleons in the center of mass of the system for the considered sym-

metric configurations. This should lead to a smoother increase in the concentration of nucleons and the density of electric charge when approaching the center of the nucleus.

The analysis of the properties of $|\Psi_0|^2$ obtained by Feynman's continual integrals method was used to refine the shell model for light nuclei [15].

The code implementing Feynman's continual integrals method was initially written for CPU. The comparison of the calculation time of the ground state energy for ^{3}He using Intel Core i5 3470 (double precision) and NVIDIA Tesla K40 (single precision) with different statistics is shown in tab. 2. Even taking into account that the code for CPU used only 1 thread, double precision and a different random number generator, the time difference is impressive. This fact allows us to increase the statistics and the accuracy of calculations in the case of using NVIDIA CUDA technology.

Table 2. Comparison of the calculation time of the ground state energy for ^{3}He nucleus

Statistics, n	Intel Core i5 3470 (1 thread, double precision), sec	NVIDIA Tesla K40 (single precision), sec	Performance gain, times
10^5	∼1854	∼8	∼232
10^6	∼18377	∼47	∼391
$5{\cdot}10^6$	-	∼221	-
10^7	-	∼439	-

The comparison of the calculation time of the square modulus of the wave function $\left|\Psi_0\left(\vec{R};\vec{r}\right)\right|^2$ for the ground state of ^{3}He using Intel Core i5 3470 and NVIDIA Tesla K40 with the statistics 10^6 for every point in the space $\left\{\vec{R};\vec{r}\right\}$ and the total number of points 43200 is shown in tab. 3. The value ∼177 days for CPU is an estimation based on the performance gain in the calculation of the ground state energy. It is evident that beside the performance gain the use of NVIDIA CUDA technology may allow us to reduce the space step in the calculation of the wave functions, as well as greatly simplify the process of debugging and testing, and in certain cases it may even enable calculations impossible before.

Table 3. Comparison of the calculation time of the square modulus of the wave function for the ground state of ^{3}He nucleus

Statistics, n	Intel Core i5 3470 (1 thread, double precision, estimation)	NVIDIA Tesla K40 (single precision)
10^6	∼177 days	∼11 hours

4. Conclusion

In this work an attempt is made to use modern parallel computing solutions to speed up the calculations of ground states of few-body nuclei by Feynman's continual integrals method. The algorithm allowing us to perform calculations directly on GPU was developed and implemented in C++ programming language. The method was applied to the nuclei consisting of nucleons, but it may also be applied to the calculation of cluster nuclei. The energy and the square modulus of the wave function of the ground states of several few-body nuclei have been calculated by

Feynman's continual integrals method using NVIDIA CUDA technology. The comparison with the square modulus of the wave function for ^{2}H calculated on CPU within the shell model was performed to confirm the correctness of the calculations. The obtained values of the theoretical binding energies are close enough to the experimental values. The theoretical charge radius and charge distribution for ^{3}He nucleus are also in good agreement with the experimental data. The results show that the use of GPGPU significantly increases the speed of calculations. This allows us to increase the statistics and the accuracy of calculations as well as reduce the space step in calculations of wave functions. It also greatly simplifies the process of debugging and testing. In certain cases the use of NVIDIA CUDA enables calculations impossible before.

The work was supported by grant 15-07-07673-a of the Russian Foundation for Basic Research (RFBR).

The paper is recommended for publication by the Program Committee of the "Parallel computational technologies (PCT) 2016" International Scientific Conference

References

1. Penionzhkevich Yu.E. Reactions Involving Loosely Bound Cluster Nuclei: Heavy Ions and New Technologies // Phys. Atom. Nucl. 2011. Vol. 74. P. 1615-1622.

2. Skobelev N.K., Penionzhkevich Yu.E., Voskoboinik E.I. et al. Fusion and Transfer Cross Sections of ^{3}He Induced Reaction on Pt and Au in Energy Range 10-24.5 MeV // Phys. Part. Nucl. Lett. 2014. Vol. 11. P. 208-215.

3. Wu Y., Ishikawa S., Sasakawa T. Three-Nucleon Bound States: Detailed Calculations of ^{3}H and ^{3}He // Few-Body Systems. 1993. Vol. 15. P. 145-188.

4. Dzhibuti R.I., Shitikova K.V. Metod gipersfericheskikh funktsiy v atomnoy i yadernoy fizike [Method of Hyperspherical Functions in Atomic and Nuclear Physics]. Moscow, Energoatomizdat, 1993. 269 P.

5. Kievsky A., Marcucci L.E., Rosati S. et al. High-Precision Calculation of the Triton Ground State Within the Hyperspherical-Harmonics Method // Few-Body Systems. 1997. Vol. 22. P. 1-10.

6. Viviani M., Kievsky A., Rosati S. Calculation of the α-Particle Ground State // Few-Body Systems. 1995. Vol. 18. P. 25-39.

7. Voronchev V.T., Krasnopolsky V.M., Kukulin V.I. A Variational Study of the Ground and Excited States of Light Nuclei in a Three-body Model on the Complete Basis. I. General Formalism // J. Phys. G. 1982. Vol. 8. P. 649-666.

8. Feynman R.P., Hibbs A.R. Quantum Mechanics and Path Integrals. New York, McGraw-Hill, 1965. 382 P.

9. Blokhintsev D.I. Osnovy kvantovoy mekhaniki [Principles of Quantum Mechanics]. Moscow, Nauka, 1976. 608 P.

10. Shuryak E.V., Zhirov O.V. Testing Monte Carlo Methods for Path Integrals in Some Quantum Mechanical Problems // Nucl. Phys. B. 1984. Vol. 242. P. 393-406.

11. Shuryak E.V. Stochastic Trajectory Generation by Computer // Sov. Phys. Usp. 1984. Vol. 27. P. 448-453.

12. Lobanov Yu.Yu. Functional Integrals for Nuclear Many-particle Systems // J. Phys. A: Math. Gen. 1996. Vol. 29. P. 6653-6669.

13. Lähde T.A., Epelbaum E., Krebs H. et al. Lattice Effective Field Theory for Medium-Mass Nuclei // Phys. Lett. B. 2014. Vol. 732. P. 110-115.

14. Borasoy B., Epelbaum E., Krebs H. et al. Lattice Simulations for Light Nuclei: Chiral Effective Field Theory at Leading Order // Eur. Phys. J. A. 2007. Vol. 31. 105-123.

15. Samarin V.V., Naumenko M.A. Study of Ground States of ^{3,4,6}He Nuclides by Feynman's Continual Integrals Method // Bull. Russ. Acad. Sci. Phys. 2016. Vol. 80, No. 3. P. 283-289.

16. NVIDIA CUDA. URL: `http://developer.nvidia.com/cuda-zone/` (accessed: 23.06.2016).

17. Sanders J., Kandrot E. CUDA by Example: An Introduction to General-Purpose GPU Programming. New York, Addison-Wesley, 2011. 290 P.

18. Perepyelkin E.E., Sadovnikov B.I., Inozemtseva N.G. Vychisleniya na graficheskikh protsessorakh (GPU) v zadachakh matematicheskoy i teoreticheskoy fiziki [Computing on Graphics Processors (GPU) in Mathematical and Theoretical Physics]. Moscow, LENAND, 2014. 176 P.

19. Ermakov S.M. Metod Monte-Karlo v vychislitel'noy matematike: vvodnyy kurs [Monte Carlo Method in Computational Mathematics. Introductory Course]. St. Petersburg, Nevskiy Dialekt, 2009. 192 P.

20. Pollyak Yu.G. Veroyatnostnoe modelirovanie na elektronnykh vychislitel'nykh mashinakh [Probabilistic Modeling on Electronic Computers]. Moscow, Sovetskoe Radio, 1971. 400 P.

21. Satcher G.R., Love W.G. Folding Model Potentials from Realistic Interaction for Heavy-Ion Scattering // Phys. Rep. 1979. Vol. 55, No. 3. P. 185-254.

22. Alvarez M.A.G., Chamon L.C., Pereira D. et al. Experimental Determination of the Ion-Ion Potential in the N=50 Target Region: A Tool to Probe Ground-State Nuclear Densities // Nucl. Phys. A. 1999. Vol. 656, No. 2. P. 187-208.

23. NRV Web Knowledge Base on Low-Energy Nuclear Physics. URL: `http://nrv.jinr.ru/` (accessed: 23.06.2016).

24. Heterogeneous Cluster of LIT, JINR. URL: `http://hybrilit.jinr.ru/` (accessed: 23.06.2016).