The Burden of Exascale

Jayadev Misra

Department of Computer Science
University of Texas at Austin

A Glimmer of Hope

"The next generation, called exaflop computers, would be capable of ...

Once thought to be just 5 or 10 years away, they now seem nearly impossible."

Superconductor Logic goes Low power
IEEE Spectrum, July 2011 (This Month), P. 18

From Petascale to Exascale
More of the same, with finer resolution

- Climate modeling, Computational biology, Code breaking, ...
- Speculative execution
- Robust computing: Computation logging/monitoring, Encrypted Computing
- Machine learning: Question answering/ Report writing
- Simulations

$ & % # _ { }

I don’t see an Exascale programming problem

, different from Petascale, Terascale, Gigascale ....

Ascale programming

The key defining characteristic is concurrency in control and data.

Concurrency

- Express concurrency explicitly or implicitly.
- Succinct representation of concurrency
  Can not enumerate threads.
- Structured Concurrency
  Fractal concurrency (Cook) for resource allocation

Static vs. Dynamic Concurrency

- Static concurrency:
  Typically synchronous parallelism. Limited range of problems.
- Dynamic concurrency:
  Typically asynchronous parallelism. Includes sequencing.

Kinds of Problems in Synchronous Parallelism

- Fast Fourier Transform
- Batcher Sort
- Ladner-Fischer Prefix sum
- Odd-Even Reductions of tridiagonal Linear Systems
- Descriptions of Recursive Connection Structures

Typical Strategy in Programming Synchronous Parallelism

- Parameterize solution by the size of the network
- Specify data movement (playing with indices)
- Specify computation at each node

Instead ...

A data structure for synchronous parallelism

- Powerlist: A list of 2^n items, $n \geq 0$.
- Smallest powerlist has a single item, $\langle x \rangle$.
- For powerlists p and q of the same length:
  (tie) $p \mid q$: p concatenated with q,
  (zip) $p \bowtie q$: interleave items from p and q, starting with p.

$$\langle 0\ 1 \rangle \mid \langle 2\ 3 \rangle = \langle 0\ 1\ 2\ 3 \rangle, \quad \langle 0\ 1 \rangle \bowtie \langle 2\ 3 \rangle = \langle 0\ 2\ 1\ 3 \rangle$$

Example of a Powerlist Function: Reverse

$$rev\langle a\ b\ c\ d\rangle = \langle d\ c\ b\ a\rangle$$

Definition of Reverse:

$$\begin{aligned} rev\langle x\rangle &= \langle x\rangle \\ rev(p \mid q) &= (rev\ q) \mid (rev\ p) \end{aligned}$$

Properties:

$$\begin{aligned} rev(p \bowtie q) &= (rev\ q) \bowtie (rev\ p) \\ rev(rev\ p) &= p \end{aligned}$$

Rotate Right and Rotate Left

$$rr\langle a\ b\ c\ d\rangle = \langle d\ a\ b\ c\rangle$$
$$rl\langle a\ b\ c\ d\rangle = \langle b\ c\ d\ a\rangle$$

$$rr\langle x\rangle = \langle x\rangle, \quad rr(u \bowtie v) = (rr\ v) \bowtie u$$

$$rl\langle x\rangle = \langle x\rangle, \quad rl(u \bowtie v) = v \bowtie (rl\ u)$$

Properties:

$$rr(rl\ p) = p$$
$$rev(rr(rev(rr\ p))) = p$$

Permutatation Function *inv*

$$
\begin{array}{lccccccccl}
 & \color{red}{000} & \color{red}{001} & \color{red}{010} & \color{red}{011} & \color{red}{100} & \color{red}{101} & \color{red}{110} & \color{red}{111} & \\
inv\langle & a & b & c & d & e & f & g & h & \rangle \quad = \\
\langle & a & e & c & g & b & f & d & h & \rangle
\end{array}
$$

$$
\begin{aligned}
inv\langle x\rangle &= \langle x\rangle \\
inv(p \mid q) &= (inv\ p) \bowtie (inv\ q)
\end{aligned}
$$

Duality Property:

$$inv(p \bowtie q) = (inv\ p) \mid (inv\ q)$$

Fast Fourier Transform: Algorithm

$$FFT\langle x\rangle = \langle x\rangle$$
$$FFT(u \bowtie v) = (U + V \times W) \mid (U - V \times W)$$

where

$$U = FFT\ u$$
$$V = FFT\ v$$
$$W = \langle \omega^0 \omega^1 \ldots \rangle$$

Message

- Implicit thread creation, manipulation
- Description is well-suited for hypercubic computation
- Narrow range of applicability

Asynchronous Parallelism

- General purpose computing with high-levels of concurrency
- Irregular problem structure (unlike synchronous parallelism)
- Thread creation, interruption, failure ... at very large scale
- Interaction with other agents, possibly in real time

Example: Map-Reduce has some of these characteristics.

What we are unable to do well

- Explicitly manage threads
- Explicitly assign threads to resources
- Explicitly specify data migration
- Explicitly integrate concurrent and sequential computing

Algebraic Approach: Orc Calculus

- Structured Concurrency
- Hierarchy, Recursion
- Implicit thread creation and manipulation

Orc Basics

- Site: Basic service or component. The value returned by a site is published.
  - add two numbers
  - decompress file
  - send an email
  - a database
  - discover a site, create a site
  - treat humans as sites
  - sites may fail
- Concurrency combinators for integrating sites.

Orc Basics; Contd.

- Theory includes nothing except the combinators.
- No notion of data type, thread, process, channel, storage, synchronization, $\cdots$
- New concepts are programmed using new sites.

Orc Calculus

- Simple Expression: just a site call, $CNN(d)$
  Publishes the value returned by the site.

- Composition of two Orc expressions:

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f >x> g$	Sequential composition
for some x from g do f	$f <x< g$	Pruning
if f halts without publishing do g	$f\ ;\ g$	Otherwise

- Definitions

Orc Calculus

- Simple Expression: just a site call, $CNN(d)$
  Publishes the value returned by the site.

- Composition of two Orc expressions:

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f > x > g$	Sequential composition
for some x from g do f	$f < x < g$	Pruning
if f halts without publishing do g	$f \; ; \; g$	Otherwise

- Definitions

Orc Calculus

- Simple Expression: just a site call, $CNN(d)$
  Publishes the value returned by the site.

- Composition of two Orc expressions:

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f > x > g$	Sequential composition
for some x from g do f	$f < x < g$	Pruning
if f halts without publishing do g	$f\ ;\ g$	Otherwise

- Definitions

Orc Calculus

- Simple Expression: just a site call, $CNN(d)$
  Publishes the value returned by the site.

- Composition of two Orc expressions:

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f > x > g$	Sequential composition
for some x from g do f	$f < x < g$	Pruning
if f halts without publishing do g	$f\ ;\ g$	Otherwise

- Definitions

Orc Calculus

- Simple Expression: just a site call, $CNN(d)$
  Publishes the value returned by the site.

- Composition of two Orc expressions:

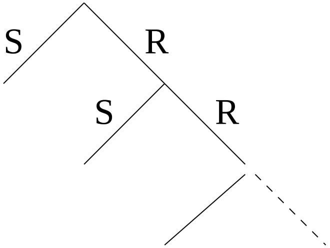
do f and g in parallel	$f \mid g$	Symmetric compositio
for all x from f do g	$f >x> g$	Sequential composition
for some x from g do f	$f <x< g$	Pruning
if f halts without publishing do g	$f \; ; \; g$	Otherwise

- Definitions

Example of a Definition: Metronome

Publish a signal every unit.

$$\textit{def}\ \ \textit{Metronome}() = \underbrace{\textit{signal}}_{S}\ \mid\ (\underbrace{\textit{Rwait}(1) \gg \textit{Metronome}()}_{R})$$



Orc language

- Adds syntactic sugar to Orc calculus
- Translated to pure Orc calculus
  All arguments in a site call are evaluated in parallel
- Mutable store only at sites

Concurrency vs. Backtracking

Given: integer `n`, list of integers `xs`
Return all subsequences of `xs` that sum to `n`.

```
sums(5,[1,-2,1,2,3]) =
 {[2, 3],[1, 1, 3],[1, -2, 1, 2, 3]}
```

`sums(5,[1,2,1])` is silent

$$def \;\; sums(0, [\,]) = \;[\,]$$

$$def \;\; sums(\_, [\,]) = \; stop$$

$$def \;\; sums(n, x : xs) = \; sums(n - x, xs) \;\; {>}ys{>} \; (x : ys) \;\; | \;\; sums(n, xs)$$

Concurrency with Maximal Parallelism

- An experiment tosses two dice.
  Experiment is a success iff sum of the two dice thrown is 7.

- $exp(n)$ runs n experiments and reports the number of successes.

$$
\begin{aligned}
\mathit{def}\ \ exp(0) &= 0 \\
\mathit{def}\ \ exp(n) &= exp(n-1) \\
&\quad + (\mathit{if}\ toss() + toss() = 7\ \mathit{then}\ 1\ \mathit{else}\ 0)
\end{aligned}
$$

- Arguments of $+$ evaluated in parallel.

Simple Parallel Auction

- A list of bidders in a sealed-bid, single-round auction.
- $b.ask()$ requests a bid from bidder b.
- Ask for bids from all bidders, then publish the highest bid.

$$\begin{aligned} &def \;\; auction([\,]) = 0 \\ &def \;\; auction(b : bs) = \max(b.ask(), auction(bs)) \end{aligned}$$

Notes:

- Arguments of max evaluated in parallel.
  All bidders are called simultaneously.
- If some bidder fails, then the auction will never complete.

Parallel Auction with Timeout

- Take a bid to be 0 if no response is received from the bidder within 8 seconds.

$$def \;\; auction([\,]) = \;\; 0$$

$$
\begin{aligned}
&def \;\; auction(b : bs) = \\
&\quad \max(\\
&\qquad b.ask() \mid (Rwait(8000) \gg 0), \\
&\qquad auction(bs) \\
&\quad)
\end{aligned}
$$

Orc Goals

- Initial Goal: Internet scripting language.
- Next: Component integration language.
- Next: A general purpose, structured "concurrent programming language".
- A very late realization: A simulation language.

$ & % # _ { }

- Avoid Technological Solutions:
  specifics of communication, topology, timing
- Avoid overspecification of control/data-flow
- Avoid mapping computations to resources until the very end

A solution is the most abstract algorithm.

Research Paradigms

- Experimentation
- Classification, Taxonomy
- Abstraction

A Philosophical Message

- Long ago: Recursion is not natural. Users will never use it.
- Today: Concurrency is not natural. Users will never get it.

Exascale programming may require combining many unnatural concepts.

And, still we may not succeed.