Will exascale computing really require new algorithms?

Kenneth (Kent) Czechowski · Aparna Chandramowlishwaran · Jee Choi · Richard (Rich) Vuduc

IPDPS'12 Panel — May 23 — Shànghǎi
Will exascale computing really require new algorithms?

**Answer:** No

*Exascale is irrelevant.* We need new algorithms because we always need new algorithms, to **improve or create new and better applications**.
Will exascale computing really require new algorithms?

Answer: No

Why? Because no useful algorithm will scale with currently projected designs.
von Neumann bottleneck

Processors become increasingly imbalanced over time.
Processors become increasingly imbalanced over time.

$$T = \max \left\{ \frac{F}{C}, \frac{Q(Z)}{\beta} \right\}$$

$$F \equiv \text{(fl)ops}$$

$$Q(Z) \equiv \text{volume of data moved}$$
Processors become increasingly imbalanced over time.

\[ T = \max \left\{ \frac{F}{C}, \frac{Q(Z)}{\beta} \right\} \]

\[= \frac{F}{C} \max \left\{ 1, \frac{C/\beta}{F/Q(Z)} \right\} \]
Processors become increasingly imbalanced over time.

\[ T = \max \left\{ \frac{F}{C}, \frac{Q(Z)}{\beta} \right\} \]

\[ = \frac{F}{C} \max \left\{ 1, \frac{C/\beta}{F/Q(Z)} \right\} \]

Examples:

\[ T_{\text{matmul}} = \frac{2n^3}{C} \max \left\{ 1, \frac{C/\beta}{\mathcal{O}(\sqrt{Z})} \right\} \]
Processors become increasingly imbalanced over time.

Examples:

\[ T \text{\ matmul} = \frac{2n^3}{C} \max \left\{ 1, \frac{C/\beta}{\Theta(\sqrt{Z})} \right\} \]

\[ T \text{\ sort} = \frac{\Theta(n \log n)}{C} \max \left\{ 1, \frac{C/\beta}{\Theta(\log Z)} \right\} \]
Cannon’s algorithm for matrix multiply

\[
T_{\text{comp}}(n; P) = \frac{2n^3}{P} \cdot \frac{1}{C}
\]

\[
\frac{T_{\text{mem}}(n; P)}{T_{\text{comp}}(n; P)} \geq \frac{C}{\beta_{\text{mem}}} \cdot \left( \frac{4}{n/\sqrt{P}} + \frac{1}{4\sqrt{2} \cdot \sqrt{Z}} \right)
\]

\[
\frac{T_{\text{net}}(n; P)}{T_{\text{comp}}(n; P)} = \frac{C}{\beta_{\text{link}}} \cdot \frac{2}{n/\sqrt{P}} \cdot \left( 1 + \frac{\alpha}{(n/\sqrt{P})^2} \right)
\]

Speed of light

Inefficiency due to communication
Cannon’s algorithm for matrix multiply

\[ T_{\text{comp}}(n; P) = \frac{2n^3}{P} \cdot \frac{1}{C} \]

\[ \frac{T_{\text{mem}}(n; P)}{T_{\text{comp}}(n; P)} \geq \frac{C}{\beta_{\text{mem}}} \left( \frac{4}{n/\sqrt{P}} + \frac{1}{4\sqrt{2} \cdot \sqrt{Z}} \right) \]

\[ \frac{T_{\text{net}}(n; P)}{T_{\text{comp}}(n; P)} = \frac{C}{\beta_{\text{link}}} \cdot \frac{2}{n/\sqrt{P}} \cdot \left( 1 + \frac{\alpha}{(n/\sqrt{P})^2} \right) \]

Inefficiency due to communication
See Aparna’s poster @ the IPDPS PhD Forum, “Communication-Optimal Parallel N-body Solvers”, and her SPAA’12 brief announcement (to appear).
Slow memory

\[ T \propto \frac{n \Delta^{3/2}}{p C_0} \left( 1 + \text{(const.)} \frac{p C_0}{\beta_{\text{mem}}} \right) \]

\[ Q(n; Z, L) = \text{words transferred} \]

Fast memory (total size = \(Z\))

\( C_0 \) op/s per core

See Aparna’s poster @ the IPDPS PhD Forum, “Communication-Optimal Parallel N-body Solvers”, and her SPAA’12 brief announcement (to appear).
See Aparna’s poster @ the IPDPS PhD Forum, “Communication-Optimal Parallel N-body Solvers”, and her SPAA’12 brief announcement (to appear).

\[ T \propto \frac{n \Delta^{3/2}}{p C_0} \left( 1 + (\text{const.}) \frac{p C_0}{\beta_{\text{mem}}} \right) \]

\[ \sim \text{digits of accuracy} \]

\[ Q(n; Z, L) = \text{words transferred} \]

\[ \alpha \text{ latency} \]
\[ \beta \text{ bandwidth} \]

\[ C_0 \text{ op/s per core} \]

\[ L \text{ words per transaction} \]

\[ \text{Fast memory (total size = } Z) \]

\[ \text{Slow memory} \]
See Aparna’s poster @ the IPDPS PhD Forum, “Communication-Optimal Parallel N-body Solvers”, and her SPAA’12 brief announcement (to appear).

\[ T \propto \frac{n\Delta^{3/2}}{pC_0} \left(1 + \text{(const.)} \frac{pC_0}{\beta_{mem}}\right) \]

- \( \Delta \) is the time per word.
- \( p \) is the number of processors.
- \( C_0 \) is the peak memory bandwidth.
- \( \beta_{mem} \) is the memory bandwidth per processor.
- \( Q(n; Z, L) \) is the number of words transferred.

\( \alpha \) is the latency and \( \beta \) is the bandwidth per transaction.

\( n \) is the number of N-body operations.

\( \Delta \) is the time per word.

\( \beta_{mem} \) is the memory bandwidth per processor.

\( C_0 \) is the peak memory bandwidth.

\( \Delta \) is the time per word.

\( \beta_{mem} \) is the memory bandwidth per processor.

\( C_0 \) is the peak memory bandwidth.

\( \Delta \) is the time per word.

\( \beta_{mem} \) is the memory bandwidth per processor.

\( C_0 \) is the peak memory bandwidth.

\( \Delta \) is the time per word.

\( \beta_{mem} \) is the memory bandwidth per processor.

\( C_0 \) is the peak memory bandwidth.
Will exascale computing really require new algorithms?

Answer: No

Why? Because algorithm designers are already working very hard to build optimal solvers. They should demand optimal machines to match!
For fixed:
* die area (transistors)
* power budget
* computation
For fixed:
* **die area** (transistors)
* **power budget**
* computation
For fixed:
* die area (transistors)
* power budget
* computation
Formal Problem

Let $n \equiv$ input size

$A \equiv$ set of algorithm variants

$M \equiv$ set of machines

$\Phi_{\text{max}} \equiv$ max system power

$\chi_{\text{max}} \equiv$ max die area

Solve $(a^*, \mu^*) = \arg \min_{a \in A, \mu \in M} T(n; a, \mu)$

subject to $\Phi(n; a, \mu) = \Phi_{\text{max}}$

$\chi(\mu) = \chi_{\text{max}}$
Let $n \equiv$ input size

$A \equiv$ set of algorithm variants

$M \equiv$ set of machines

$\Phi_{\text{max}} \equiv$ max system power

$\chi_{\text{max}} \equiv$ max die area

Solve $(a^*, \mu^*) = \arg\min_{a \in A, \mu \in M} T(n; a, \mu)$

subject to $\Phi(n; a, \mu) = \Phi_{\text{max}}$

$\chi(\mu) = \chi_{\text{max}}$
Sample Projection (2018)
1 processor

MatMult

3D FFT

Cache size (MiB)
Memory bandwidth (TB/s)

Sample Projection (2018)

1 processor


Wednesday, May 23, 12
Sample Projection (2018)
1 processor

MatMult

3D FFT


Wednesday, May 23, 12
Sample Projection (2018)

1 processor

MatMult

3D FFT

Evolution ...

Echelon

GPU

CPU

Sample Projection (2018)

1 processor

MatMult

3D FFT

Cache size (MiB)

Memory bandwidth (TB/s)


Revolution?

< \( \frac{2}{3} \times \)

\(~ 2 \times\)
The diagram illustrates the performance of different operations (MatMult, 3D FFT) on a single processor, with the cache size on the x-axis and memory bandwidth on the y-axis. The contours represent different values of a parameter, indicating trade-offs between cache size and memory bandwidth for achieving optimal performance. The diagram assumes the work by Keckler et al. (2011) as cited.

A single die configuration (cores v. LLC) might be fine …


... if we could get “extreme” reconfigurability of power.
Solutions to the constrained optimization problem

Matrix multiply — 8 EF/s peak (1M nodes)

<table>
<thead>
<tr>
<th>Transistor Budget</th>
<th>Node Power Budget</th>
<th>System Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Die Area (mm^2)</td>
<td>Node Power (W)</td>
<td>System Power (MW)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>2.8</td>
<td>0.1</td>
</tr>
<tr>
<td>35</td>
<td>2.8</td>
<td>0.1</td>
</tr>
<tr>
<td>71</td>
<td>5.6</td>
<td>0.3</td>
</tr>
<tr>
<td>71</td>
<td>5.6</td>
<td>0.3</td>
</tr>
<tr>
<td>106</td>
<td>8.3</td>
<td>0.6</td>
</tr>
<tr>
<td>106</td>
<td>8.3</td>
<td>0.6</td>
</tr>
<tr>
<td>141</td>
<td>11.1</td>
<td>1.0</td>
</tr>
<tr>
<td>141</td>
<td>11.1</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Relative to notional Echelon:
~ 5x faster for MM
~ 0.9x as fast for 3D FFT

3D FFT — 230 PF/s peak (4k nodes)

<table>
<thead>
<tr>
<th>Transistor Budget</th>
<th>Node Power Budget</th>
<th>System Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Die Area (mm^2)</td>
<td>Node Power (W)</td>
<td>System Power (MW)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>5</td>
<td>0.1</td>
</tr>
<tr>
<td>35</td>
<td>5</td>
<td>0.1</td>
</tr>
<tr>
<td>71</td>
<td>10</td>
<td>0.3</td>
</tr>
<tr>
<td>71</td>
<td>10</td>
<td>0.3</td>
</tr>
<tr>
<td>106</td>
<td>15</td>
<td>0.6</td>
</tr>
<tr>
<td>106</td>
<td>15</td>
<td>0.6</td>
</tr>
<tr>
<td>141</td>
<td>20</td>
<td>1.0</td>
</tr>
<tr>
<td>141</td>
<td>20</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Relative to notional Echelon:
~ 28x faster for 3D FFTs
~ 0.14x as fast for MM
In summary, we need new algorithms in spite of—not because of—the hardware we may get at exascale.

The question is whether we can articulate what algorithms really need, in terms of accuracy and performance, in a way that will lead to better systems.