Lightweight Requirements Engineering for Exascale Co-design

Felix Wolf, TU Darmstadt
Acknowledgement

• Alexandru Calotoiu, TU Darmstadt
• Alexander Graf, TU Darmstadt
• Torsten Hoefler, ETH Zurich
• Daniel Lorenz, TU Darmstadt
• Sergei Shudler, TU Darmstadt
• Sebastian Rinke, TU Darmstadt
Co-design

Workload

System

Better algorithms
Current performance might be deceptive…
Hardware-specific performance models

Application 1

- Performance model 1.1
- Performance model 1.2
- Performance model 1.3

System 1

System 2

...
Application-centric requirements models

System 1

System 2

System n

Application 1

Requirements model 1
Data metabolism at the hardware / software interface
Hardware-independent requirement metrics
Requirements model of an application

Set of functions $r_i(p,n)$

with each $r_i$ representing one of the requirement metrics

- All metrics refer to single process
- We model **neither time nor energy**
Lightweight requirements engineering for (exascale) co-design

- Collect portable requirement metrics
- Derive requirement models
- Extrapolate to new system
## Collection of requirement metrics

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Metric</th>
<th>Profiling tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td># Floating-point operations</td>
<td><img src="#" alt="Score-P" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="#" alt="PAPI" /></td>
</tr>
<tr>
<td>Network comm.</td>
<td># Bytes sent &amp; received</td>
<td><img src="#" alt="Score-P" /></td>
</tr>
<tr>
<td>Memory footprint</td>
<td># Bytes used</td>
<td>getrusage()</td>
</tr>
<tr>
<td>Memory access</td>
<td># Loads &amp; stores</td>
<td><img src="#" alt="Score-P" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="#" alt="PAPI" /></td>
</tr>
<tr>
<td>Memory locality</td>
<td>Stack distance</td>
<td><img src="#" alt="Threadspotter" /></td>
</tr>
</tbody>
</table>

Collection single-threaded (#FLOPS roughly independent of #threads)
Modeling locality

Reuse distance vs. stack distance

Paratools Threadspotter

Reuse distance = 1
Stack distance = 1

Reuse distance = 3
Stack distance = 2
Automatic empirical performance modeling with Extra-P

```
main() {
    foo()
    bar()
    compute()
}
```

Instrumentation

Small-scale measurements

Extra-P

Input

Output

Human-readable, multi-parameter performance models

$$f(x_1, \ldots, x_m) = \sum_{k=1}^{n} c_k \prod_{l=1}^{m} x_{kl}^{i_{kl}} \cdot \log_2^{j_{kl}}(x_l)$$


www.scalasca.org/software/extra-p/download.html
Test applications

Kripke

MILC

LULESH

icoFoam

Relearn
Experimental setup

JUQUEEN
@ Jülich Supercomputing Centre
IBM Blue Gene/Q

Lichtenberg @ TU Darmstadt
Intel Xeon with Infiniband
### Modeling application requirements

Models represent **per-process** effects

- $p$ – number of processes
- $n$ – problem size per process

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Metric</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>#FLOPs</td>
<td>$10^5 \cdot n \cdot \log(n) \cdot p^{0.25} \cdot \log(p)$</td>
</tr>
<tr>
<td>Communication</td>
<td>#Bytes sent &amp; received</td>
<td>$10^3 \cdot n \cdot p^{0.25} \cdot \log(p)$</td>
</tr>
<tr>
<td>Memory access</td>
<td>#Loads &amp; stores</td>
<td>$10^5 \cdot n \cdot \log(n) \cdot \log(p)$</td>
</tr>
<tr>
<td>Memory footprint</td>
<td>#Bytes used</td>
<td>$10^5 \cdot n \cdot \log(n)$</td>
</tr>
<tr>
<td>Memory locality</td>
<td>Stack distance</td>
<td>Constant</td>
</tr>
</tbody>
</table>

*Figures are hypothetical and used for illustrative purposes.*
Determining requirements on a new system

Available sockets → # Processes → Overall problem size → Requirement models

Available memory per process → Problem size per process → Requirements #FLOPS #Bytes sent ...

- # Processes
- Overall problem size
- Requirements #FLOPS #Bytes sent ...
- Available sockets
- Available memory per process
- Requirement models
Requirements engineering process

- Memory capacity
- Memory bandwidth
- Computational performance
- Network bandwidth
Case study
Three system upgrades

- Racks x 2
- Sockets x 2
- Memory x 2
<table>
<thead>
<tr>
<th>Ratios</th>
<th>Apps</th>
<th>Kripke</th>
<th>LULESH</th>
<th>MILC</th>
<th>Relearn</th>
<th>icoFoam</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Upgrade A:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem size per process</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>Overall problem size</td>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Computation</td>
<td></td>
<td>1</td>
<td>1.2</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>Communication</td>
<td></td>
<td>1</td>
<td>1.2</td>
<td>1</td>
<td>1</td>
<td>0.7</td>
<td>1</td>
</tr>
<tr>
<td>Memory accesses</td>
<td></td>
<td>2</td>
<td>1.2</td>
<td>2.8</td>
<td>2</td>
<td>0.7</td>
<td>1</td>
</tr>
<tr>
<td><strong>System Upgrade B:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem size per process</td>
<td></td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
<td>0.5</td>
</tr>
<tr>
<td>Overall problem size</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
<td>0.6</td>
<td>1</td>
</tr>
<tr>
<td>Computation</td>
<td></td>
<td>0.5</td>
<td>0.6</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
<td>0.5</td>
</tr>
<tr>
<td>Communication</td>
<td></td>
<td>0.5</td>
<td>0.6</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
<td>0.5</td>
</tr>
<tr>
<td>Memory accesses</td>
<td></td>
<td>0.5</td>
<td>1</td>
<td>1.4</td>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>System Upgrade C:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem size per process</td>
<td></td>
<td>2</td>
<td>1.4</td>
<td>2</td>
<td>4</td>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>Overall problem size</td>
<td></td>
<td>2</td>
<td>1.4</td>
<td>2</td>
<td>4</td>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>Computation</td>
<td></td>
<td>2</td>
<td>1.4</td>
<td>2</td>
<td>4</td>
<td>1.7</td>
<td>2</td>
</tr>
<tr>
<td>Communication</td>
<td></td>
<td>2</td>
<td>1.4</td>
<td>2</td>
<td>4</td>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>Memory accesses</td>
<td></td>
<td>2</td>
<td>1.4</td>
<td>2</td>
<td>4</td>
<td>1.4</td>
<td>2</td>
</tr>
</tbody>
</table>
### Case study II
Three exascale strawman systems

<table>
<thead>
<tr>
<th>Metric</th>
<th>Massively parallel</th>
<th>Vector</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>$2 \times 10^4$</td>
<td>$5 \times 10^4$</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Processors</td>
<td>$2 \times 10^9$</td>
<td>$5 \times 10^7$</td>
<td>$10^8$</td>
</tr>
<tr>
<td>Processors per node</td>
<td>$10^5$</td>
<td>$10^3$</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Memory per processor</td>
<td>$5 \times 10^6$</td>
<td>$2 \times 10^8$</td>
<td>$10^8$</td>
</tr>
<tr>
<td>Flop/s per processor</td>
<td>$5 \times 10^8$</td>
<td>$2 \times 10^{10}$</td>
<td>$10^{10}$</td>
</tr>
</tbody>
</table>

Many but weak processors
Few but powerful processors
Moderate number of moderate processors

Total memory: 10 PB
## Case study II
### Three exascale strawman systems

<table>
<thead>
<tr>
<th>Metric</th>
<th>Massively parallel</th>
<th>Vector</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kripke</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum overall problem size</td>
<td>$10^{10}$</td>
<td>$10^{10}$</td>
<td>$10^{10}$</td>
</tr>
<tr>
<td>Minimum wall time for benchmark problem [s]</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td><strong>Lulesh</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum overall problem size</td>
<td>$3.9 \cdot 10^{10}$</td>
<td>$1.7 \cdot 10^{10}$</td>
<td>$1.9 \cdot 10^{10}$</td>
</tr>
<tr>
<td>Minimum wall time for benchmark problem [s]</td>
<td>40</td>
<td>21.5</td>
<td>33</td>
</tr>
<tr>
<td><strong>MILC</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum overall problem size</td>
<td>$10^{10}$</td>
<td>$10^{10}$</td>
<td>$10^{10}$</td>
</tr>
<tr>
<td>Minimum wall time for benchmark problem [s]</td>
<td>$10^2$</td>
<td>$10^2$</td>
<td>$10^2$</td>
</tr>
<tr>
<td><strong>Relearn</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum overall problem size</td>
<td>$5 \cdot 10^{10}$</td>
<td>$4 \cdot 10^{12}$</td>
<td>$10^{12}$</td>
</tr>
<tr>
<td>Minimum wall time for benchmark problem [s]</td>
<td>4</td>
<td>0.02</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Bigger problem versus faster solution

Vector system clear winner
Summary

Application-centric requirements models
- No need to integrate hardware knowledge
- Generation via standard profiling tools
- Memory locality taken into account

Practical co-design process
- Extrapolates requirements to envisaged system
- Points out bottlenecks on both sides

Automated BOE co-design for large workloads
Tasking

Idea – separate problem decomposition from concurrency

• Decompose problem into a set of tasks and insert them into task pool
• Threads fetch them from there until all tasks are completed and task pool empty. Note that a task may create new tasks
• Advantage: good load balance if problem is over-decomposed
Tasking (2)

- Task-based paradigms: Cilk, OmpSs, OpenMP,…
- Scheduling managed by the runtime system
- Example:

```c
#pragma omp task shared(x)
x = fib(n - 1);
#pragma omp task shared(y)
y = fib(n - 2);
#pragma omp taskwait
return x + y;
```
Efficiency of task-based applications – performance issues

- Input size
- Task graph
- Core count

\[
\text{const. efficiency} = \frac{S}{p}
\]
Efficiency of task-based applications – performance issues (2)

- Input size
- Task graph
- Core count

const. efficiency $= \frac{S}{p}$

[Diagram showing a tree-like task graph with nodes and edges, and a symbol for input size and core count.]
Efficiency of task-based applications – performance issues (3)

Input size

Core count

Resource contention

const. efficiency = \frac{S}{p}
Task dependency graph (TDG)

- Nodes – tasks, edges – dependencies
- \( p, n \) – processing elements, input size
- \( T_1(n) \) – all the task times (work)
- \( T_\infty(n) \) – longest path (depth)
- \( \pi(n) = \frac{T_1(n)}{T_\infty(n)} \) – average parallelism
- \( T_p(n) \) – execution time
- \( S_p(n) = \frac{T_1(n)}{T_p(n)} \) – speedup

\[ T_1 = 45 \]
\[ T_\infty = 25 \]
Efficiency & isoeficiency

- Efficiency is defined as: $E(p,n) = \frac{S_p(n)}{p} \leq \min\left\{1, \frac{\pi(n)}{p}\right\} = E_{ub}(p,n)$

- Isoefficiency binds together the core count and the input size for a specific, constant efficiency: $n = f_E(p)$
  - A contour line on the efficiency surface

- Example: Mergesort
  - $\pi(n) = \log n$
  - Surface depicts $E_{ub}(p,n)$
Modeled efficiency functions

\[ E_{ub}(p,n) \] – upper bound based on avg. parallelism

\[ \Delta_{str} = E_{ub}(p,n) - E_{cf}(p,n) \]

\[ E_{cf}(p,n) \] – contention-free replays

\[ \Delta_{con} = E_{cf}(p,n) - E_{ac}(p,n) \]

\[ E_{ac}(p,n) \] – reflects actual performance

**Structural discrepancy:** characterizes optimization potential on the task-graph level

**Contention discrepancy:** shows how severe the resource contention is
Co-design aspects

<table>
<thead>
<tr>
<th>App.</th>
<th>Model</th>
<th>Input size for $p = 60$, $E = 0.8$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$E_{ac} = 0.98 - 5.11 \times 10^{-3} p^{1.25} + 1.76 \times 10^{-3} p^{1.25} \log n$</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>$E_{cf} = 0.97 - 1.46 \times 10^{-2} p^{1.25} + 9.26 \times 10^{-3} p^{1.25} \log n$</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>$E_{ub} = \min \left{ 1, \left( 25.48 + 0.49 n^{2.75} \log n \right) p^{-1} \right}$</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>$E_{ac} = 1.55 - 1.02 p^{0.25} + 4.59 \times 10^{-2} p^{0.25} \log n$</td>
<td>83,600 x 83,600</td>
</tr>
<tr>
<td></td>
<td>$E_{cf} = 1.26 - 0.65 p^{0.33} + 3.89 \times 10^{-2} p^{0.33} \log n$</td>
<td>12,680 x 12,680</td>
</tr>
<tr>
<td></td>
<td>$E_{ub} = \min \left{ 1, \left( 0.25 n^{0.75} \right) p^{-1} \right}$</td>
<td>1,200 x 1,200</td>
</tr>
</tbody>
</table>

For example (Strassen): $E_{ac} = 1.55 - 1.02 p^{0.25} + 4.59 \times 10^{-2} p^{0.25} \log n$

Let $E = 0.8$ and $p = 60$: $0.8 = 1.55 - 1.02 \cdot 60^{0.25} + 4.59 \times 10^{-2} \cdot 60^{0.25} \log n$

After solving: $n = 83,600$
Extra-P 3.0

- GUI improvements, better stability, additional features
- Tutorials available through VI-HPS and upon request

http://www.scalasca.org/software/extra-p/download.html
## Related publications

<table>
<thead>
<tr>
<th></th>
<th>Authors</th>
<th>Title</th>
<th>Conference/Event</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
</table>
Thank you!