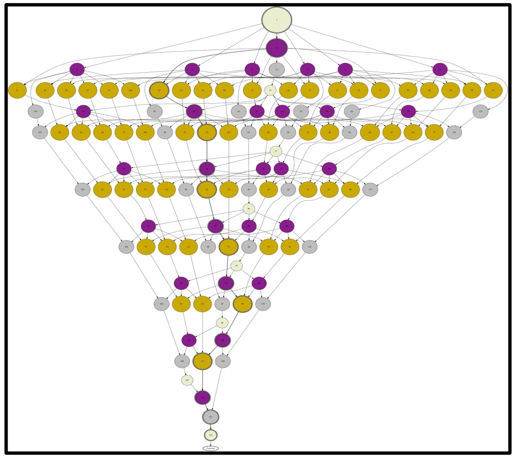
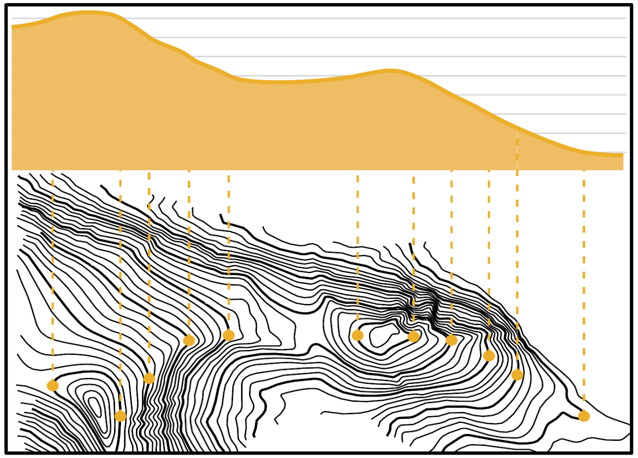
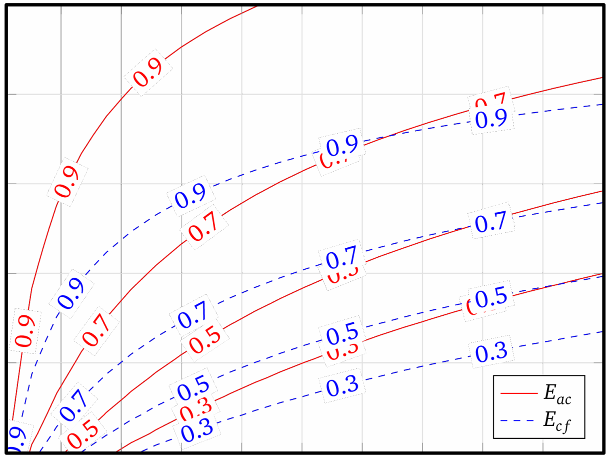
Isoefficiency in Practice: Configuring and Understanding the Performance of Task-based Applications*

Sergei Shudler[1], **Alexandru Calotoiu**[1], **Torsten Hoefler**[2], **Felix Wolf**[1]







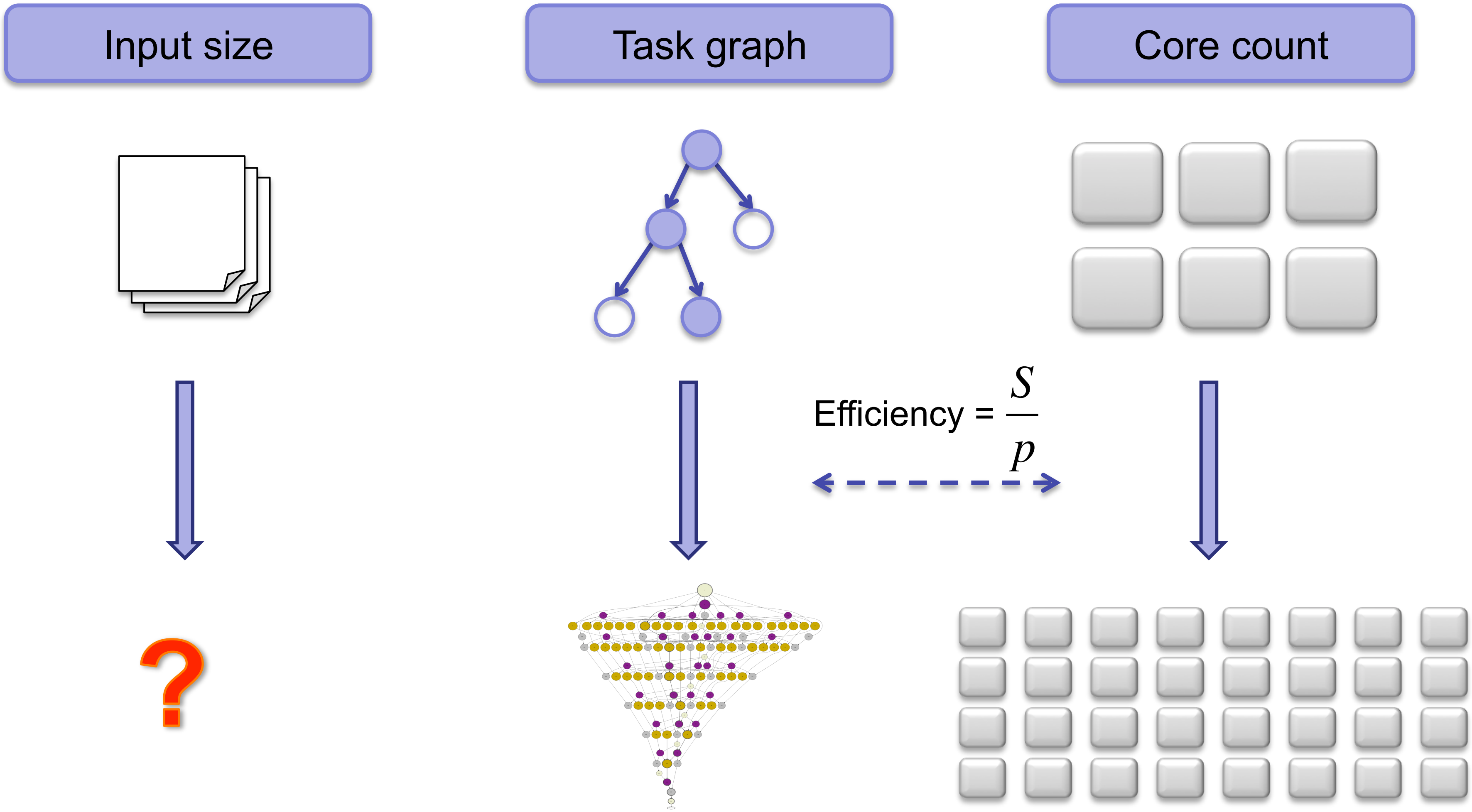
[1] TU Darmstadt, [2] ETH Zürich

* Published in PPoPP '17

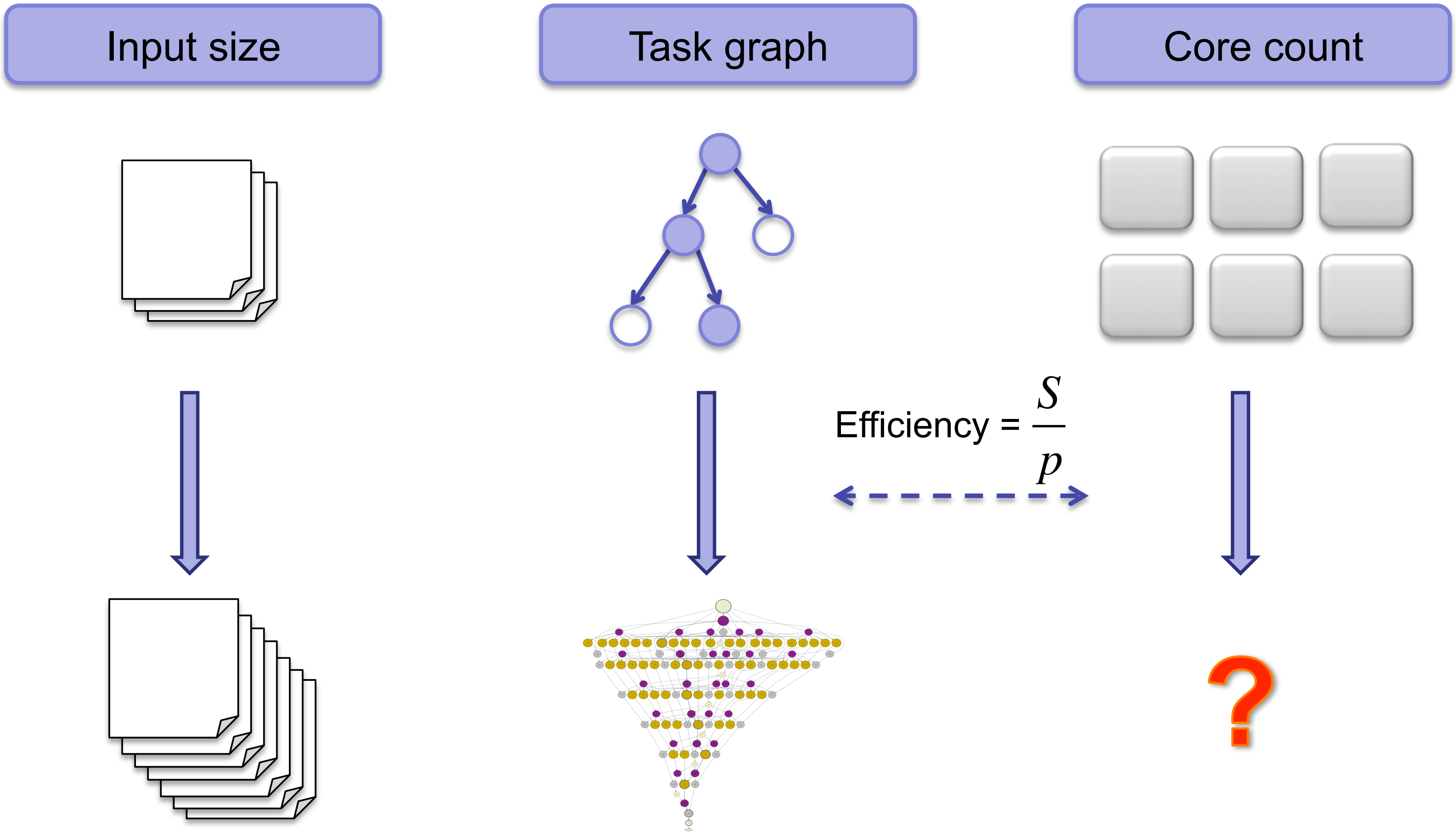
Agenda

- Overview
- Task dependency graph analysis
- Isoefficiency modeling
- Evaluation
- Conclusion

Efficiency of task-based applications – performance issues (1)

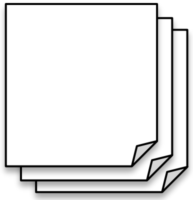


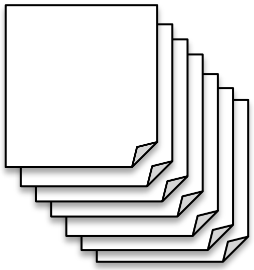
Efficiency of task-based applications – performance issues (2)



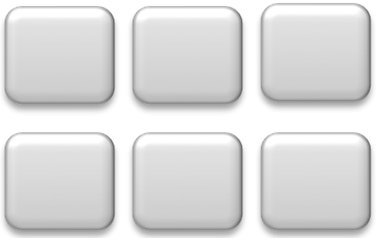
Efficiency of task-based applications – performance issues (3)

Input size





Core count



Efficiency = $\frac{S}{p}$

Resource contention

Addressed questions

TECHNISCHE UNIVERSITÄT DARMSTADT

Fundamental scalability limitations in a task-based program

Core count for a given input size

?

Poor scaling caused by resource contention overhead

Input size for a given core count

Further optimization potential: dependencies, scheduling, granularity

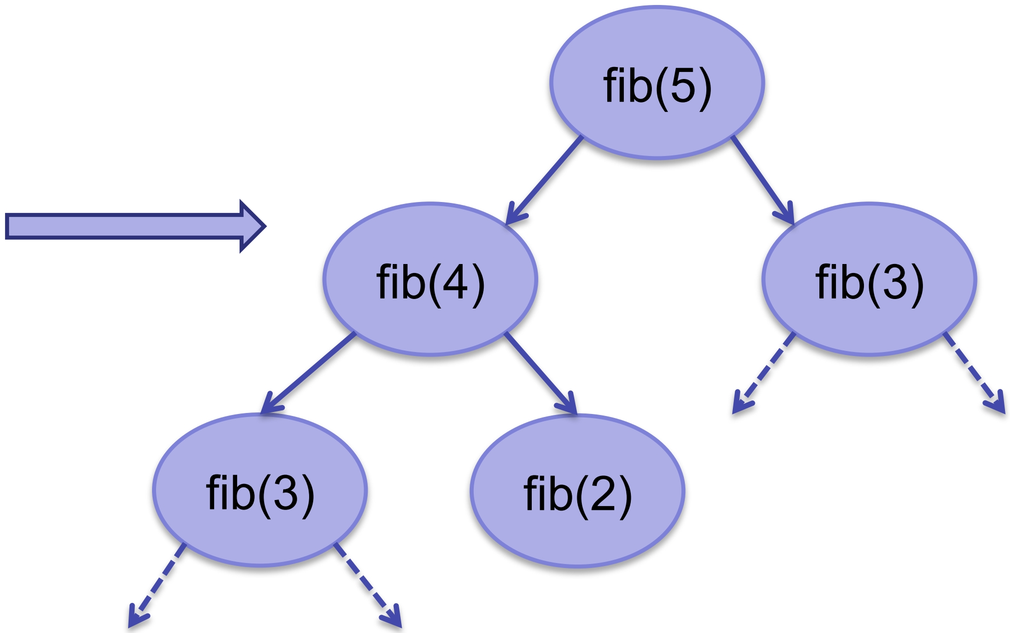
Agenda

- Overview
- Task dependency graph analysis
- Isoefficiency modeling
- Evaluation
- Conclusion

Task-based programs

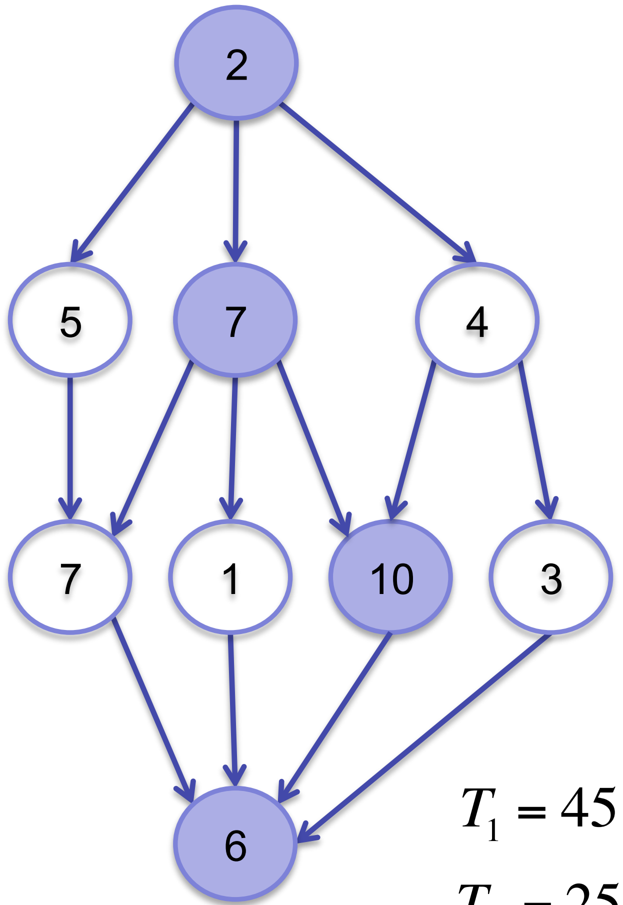
- Task-based paradigms: Cilk, OmpSs, OpenMP,…
- Scheduling managed by the runtime system
- Example:

```
#pragma omp task shared(x)
x = fib( n – 1 );
#pragma omp task shared(y)
y = fib( n – 2 );
#pragma omp taskwait
return x + y;
```



Task dependency graph (TDG)

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



$T_1 = 45$

$T_\infty = 25$

TDG rules

TECHNISCHE UNIVERSITÄT DARMSTADT

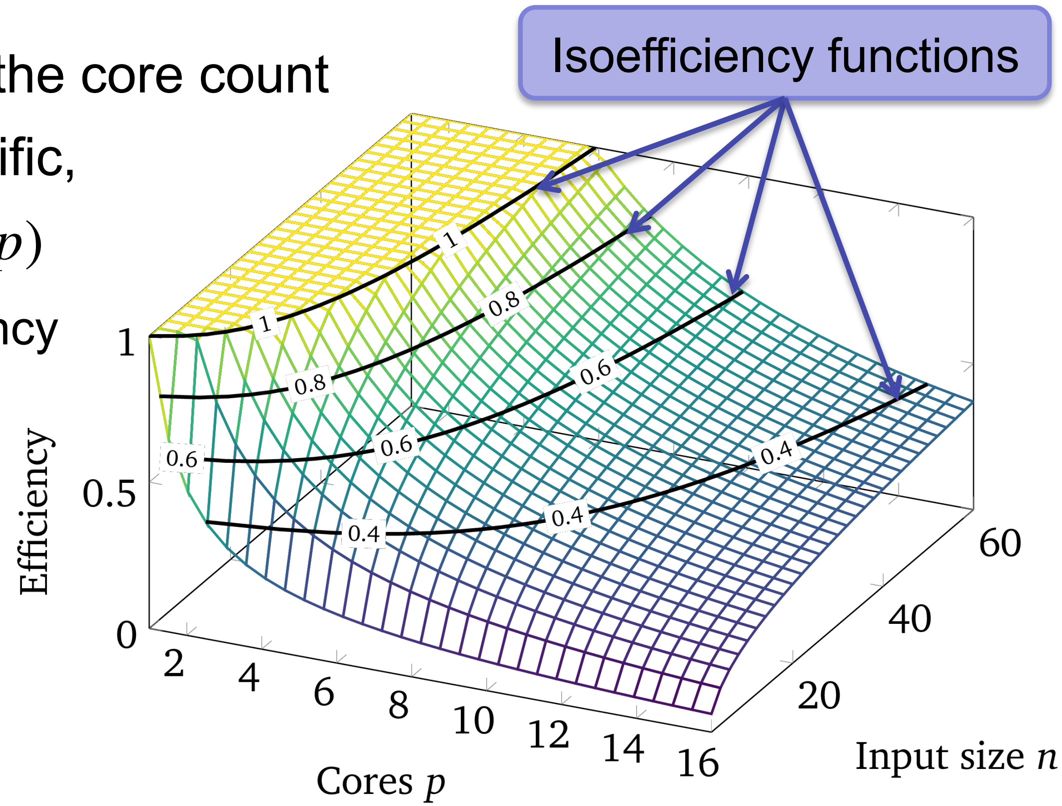
- Brent's lemma: $T_p(n) \le \frac{T_1(n) - T_\infty(n)}{p} + T_\infty(n)$
- Work rule: $T_p(n) \ge \frac{T_1(n)}{p}$ or: $S_p(n) \le p$
  - Ignore super-linear speedups for simplicity
- Depth rule: $T_p(n) \ge T_\infty(n)$ or: $S_p(n) \le \pi(n)$
  - Cannot execute faster than the critical path
- In summary: $S_p(n) \le \min\{p, \pi(n)\}$

Agenda

- Overview
- Task dependency graph analysis
- **Isoefficiency modeling**
- Evaluation
- Conclusion

Efficiency & isoefficiency

- 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 $E_{ub}(p,n)$



Solution: Modeling (iso)efficiency functions

TECHNISCHE UNIVERSITÄT DARMSTADT

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

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

Structural discrepancy: characterizes the optimization potential

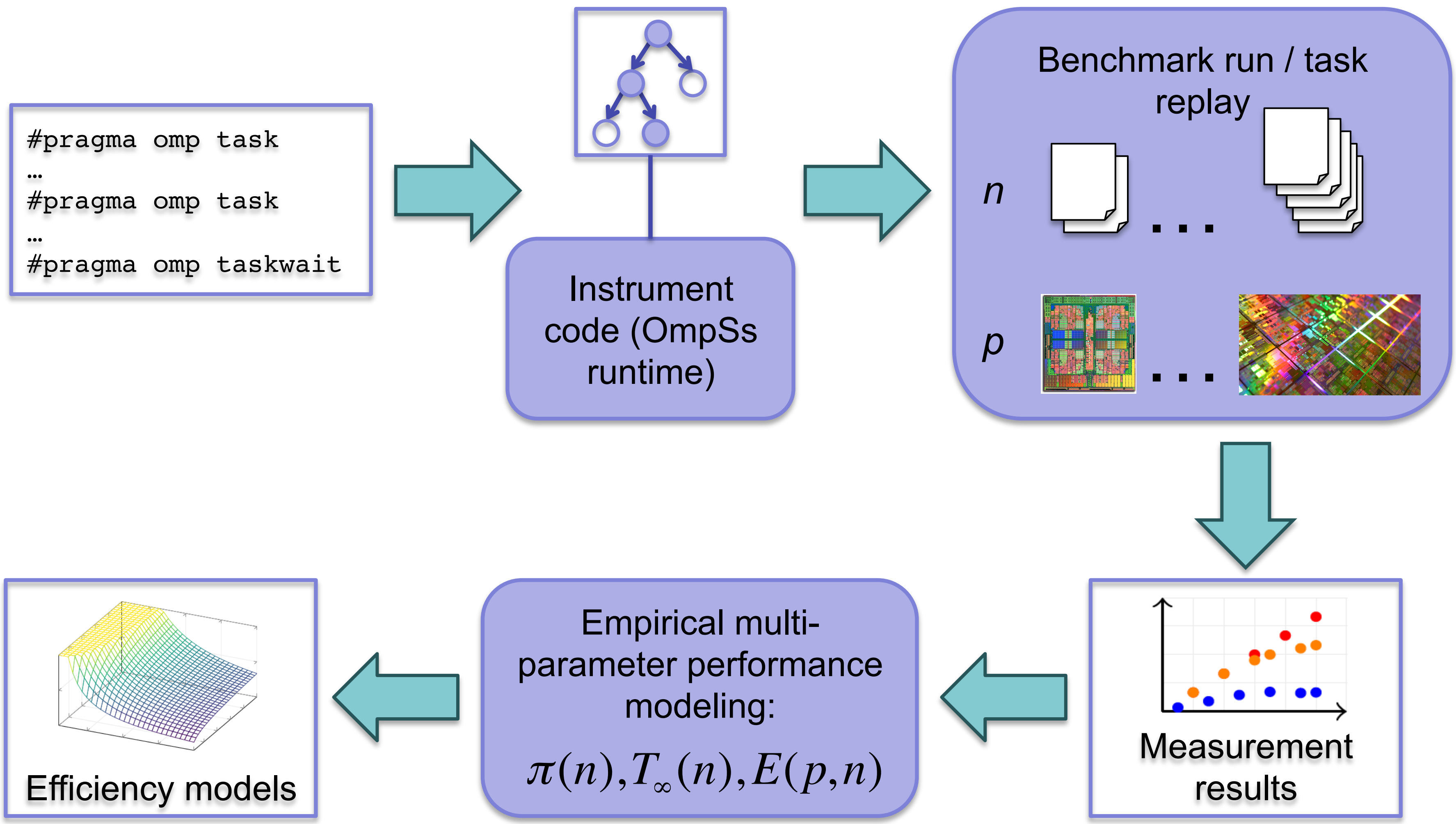
$E_{cf}(p,n)$ – contention-free replays

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

Contention discrepancy: shows how severe the resource contention is

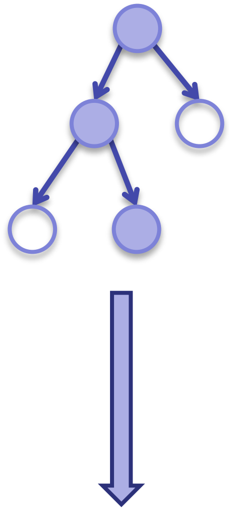
$E_{ac}(p,n)$ – reflects realistic performance

Modeling workflow



Contention-free replay engine

- Uses OmpSs runtime API
- Replay on multiple threads
- No actual code execution (busy-waiting)
- Respects dependencies
- Same scheduling policy
- Minimum memory accesses

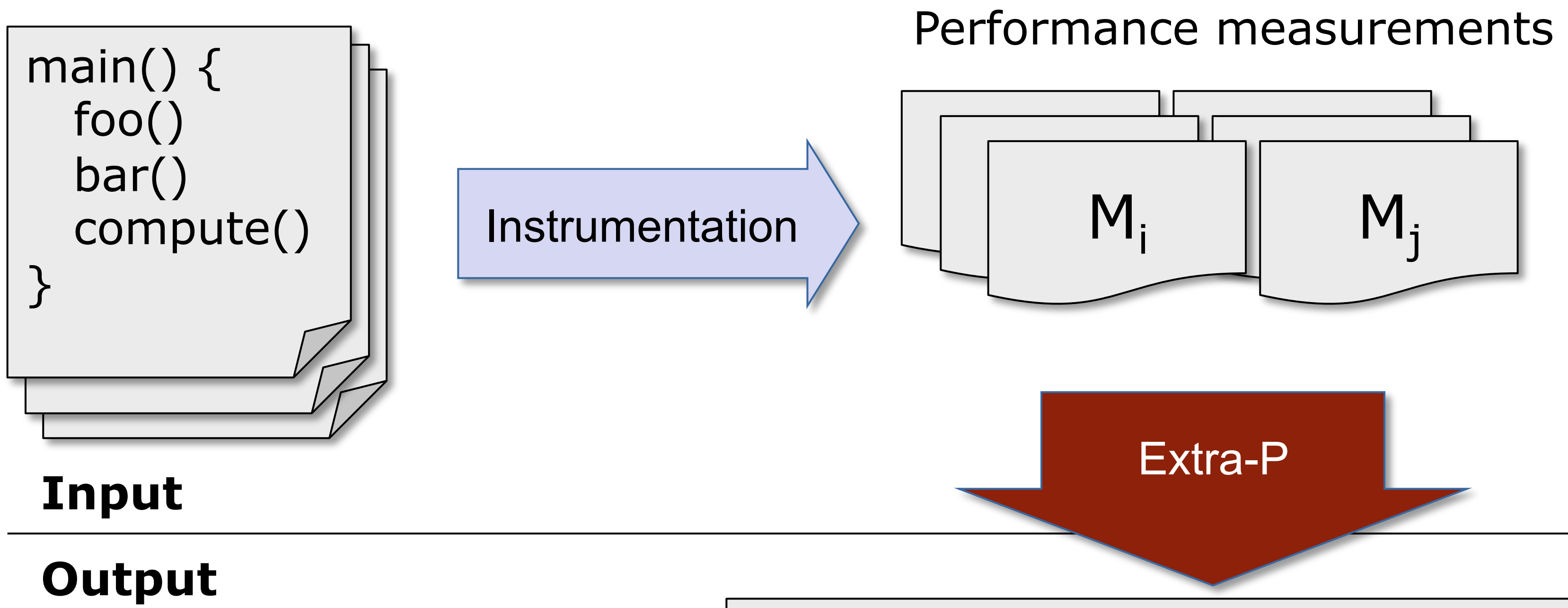


```
void exec_task( double t )
{
   double t_c = … //curr time
   double t_e = t_c + t;
   while( t_c < t_e )
      t_c = … //curr time
}

//…

nanos_create_wd_compact(&exec_task)
```

Performance modeling with Extra-P



Human-readable, multi-parameter performance models of all functions

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

A. Calotoiu, et al.: Fast Multi-Parameter Performance Modeling (*CLUSTER '16*)

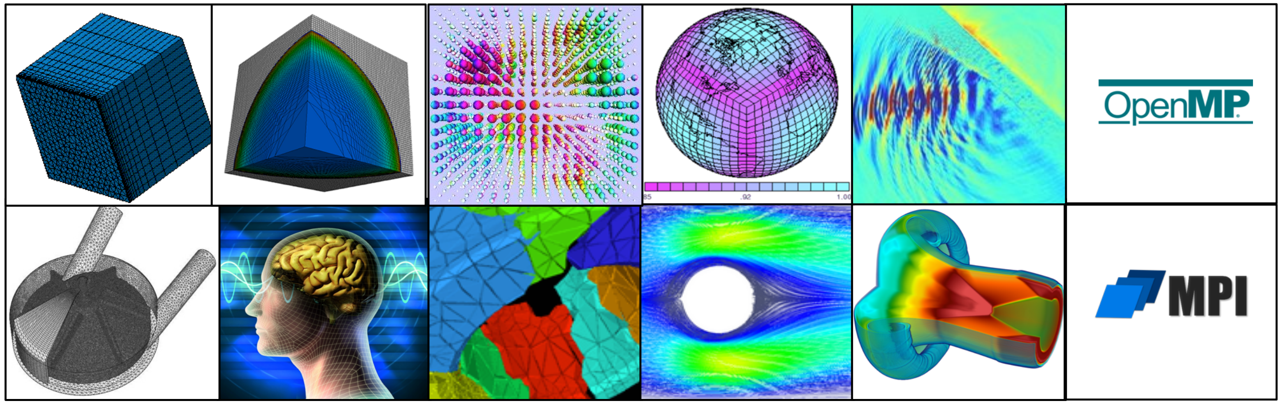
Extra-P

TECHNISCHE UNIVERSITÄT DARMSTADT

Software

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

Case studies



Agenda

- Overview
- Task dependency graph analysis
- Isoefficiency modeling
- Evaluation
- Conclusion

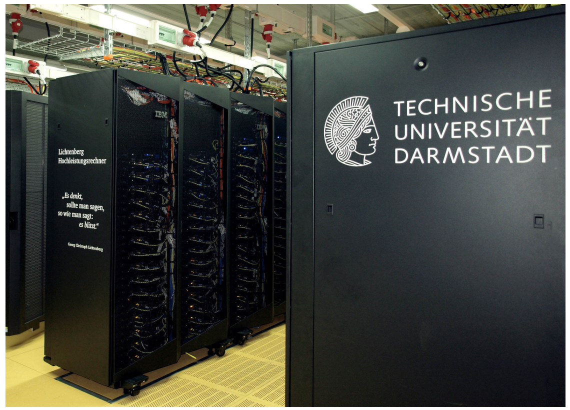
Experiments setup

- Barcelona OpenMP Task Suite (BOTS) and Barcelona Application Repository (BAR)
  - Cholesky, FFT, Fib, NQueens, Sort, SparseLU, Strassen

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

BSC Application Repository

- NUMA node with four Intel Xeon E7-4890 v2 processors (Ivy Bridge)
  - 60 cores in total

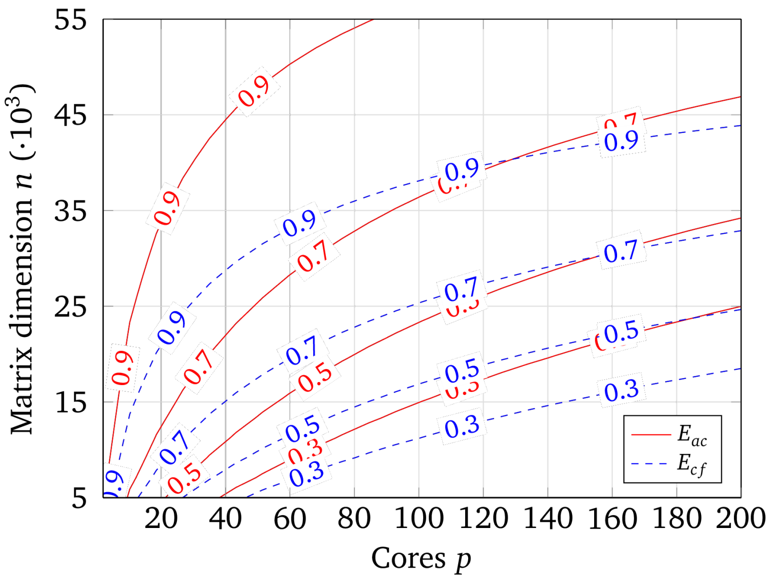


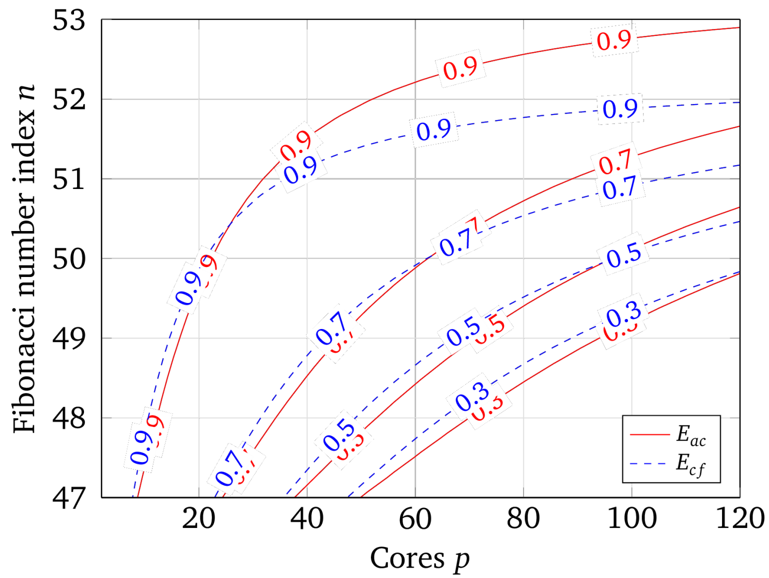
Depth and average parallelism models (excerpt)

Application (origin)	$T_\infty(n)$	$\pi(n)$
Cholesky (BAR)	$O(n^{1.75}\log n)$	$O(n)$
FFT (BAR)	$O(n^{2.75}\log n)$	$O(n^{0.67}\log n)$
Nqueens (BOTS)	$O(n^2\log n)$	$O(n^{2.875}\log n)$
Sort (BOTS)	$O(\sqrt{n})$	$O(\sqrt{n})$
SparseLU (BAR)	$O(n^{0.75}\log n)$	$O(n^{1.75}\log n)$
Strassen (BOTS)	$O(n^2\log n)$	$O(n^{0.75})$

$T_\infty(n)$ grows faster or as fast as $\pi(n)$

Efficiency & isoefficiency models (excerpt)

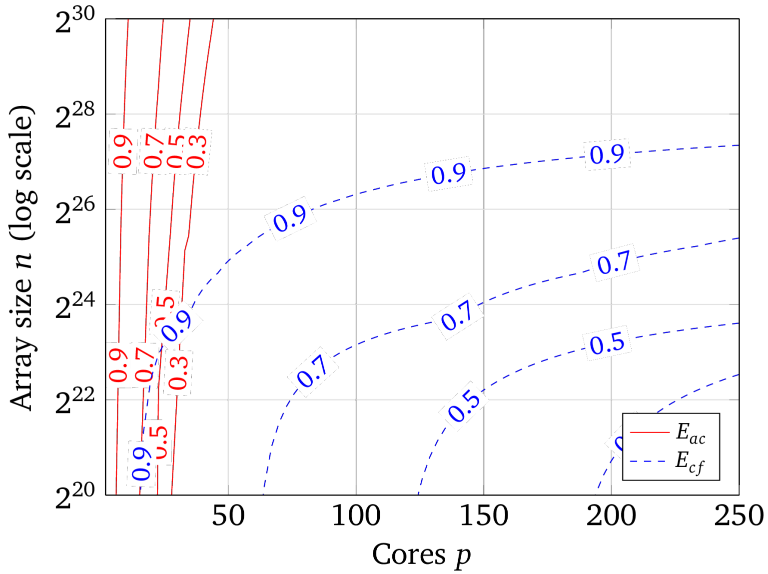


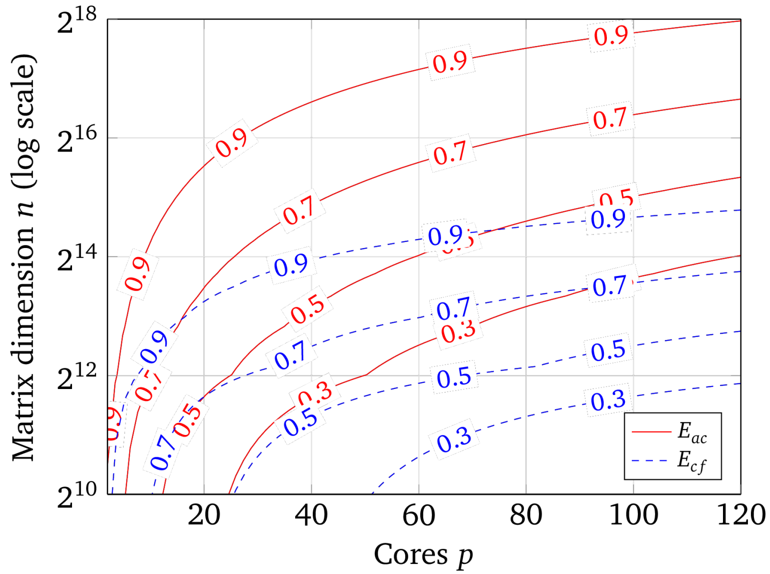


Cholesky models	Fibonacci models
$E_{ac} = 1.09 - 0.51\sqrt{p} + 3.11 \cdot 10^{-2}\sqrt{p}\log n$	$E_{ac} = 0.98 - 5.11 \cdot 10^{-3} p^{1.25} + 1.76 \cdot 10^{-3} p^{1.25} \log n$
$E_{cf} = 1.14 - 0.54\sqrt{p} + 3.4 \cdot 10^{-2}\sqrt{p}\log n$	$E_{cf} = 0.97 - 1.46 \cdot 10^{-2} p^{1.25} + 9.26 \cdot 10^{-3} p^{1.25} \log n$
$E_{ub} = \min\left\{1, \left(2.29 + 2.35 \cdot 10^{-3} n\right) p^{-1}\right\}$	$E_{ub} = \min\left\{1, \left(25.48 + 0.49 n^{2.75} \log n\right) p^{-1}\right\}$

$C - Af(p) + Bf(p)g(n)$ → C: max, -A*f*(*p*): reduction, B*f*(*p*)*g*(*n*): gain

Efficiency & isoefficiency models (excerpt)





Sort models	Strassen models
$E_{ac} = 0.99 - 9.2\cdot 10^{-3} p^{1.5} + 2.29\cdot 10^{-4} p^{1.5} \log n$	$E_{ac} = 1.55 - 1.02 p^{0.25} + 4.59\cdot 10^{-2} p^{0.25} \log n$
$E_{cf} = 1.0 - 4.61\cdot 10^{-2} p^{0.75} + 1.62\cdot 10^{-3} p^{0.75} \log n$	$E_{cf} = 1.26 - 0.65 p^{0.33} + 3.89\cdot 10^{-2} p^{0.33} \log n$
$E_{ub} = \min\left\{1, \left(3.53 + 3.32\cdot 10^{-2}\sqrt{n}\right) p^{-1}\right\}$	$E_{ub} = \min\left\{1, \left(0.25 n^{0.75}\right) p^{-1}\right\}$

$C - Af(p) + Bf(p)g(n)$ → C: max, -A*f*(*p*): reduction, B*f*(*p*)*g*(*n*): gain

Co-Design aspects

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

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

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

After solving: $n = 83{,}600$

Agenda

- Overview
- Task dependency graph analysis
- Isoefficiency modeling
- Evaluation
- Conclusion

Conclusion

TECHNISCHE UNIVERSITÄT DARMSTADT

- Practical way to use isoefficiency
- Modeling of resource contention overhead
- Uncover hidden parallelism potential
- Co-design: derive input sizes for future machines

References (partial):

S. Shudler et al.: Isoefficiency in Practice: Configuring and Understanding the Performance of Task-based Applications (*PPoPP'17*)

A. Calotoiu et al.: Fast Multi-Parameter Performance Modeling (*CLUSTER '16*)

S. Shudler et al.: Exascaling Your Library: Will Your Implementation Meet Your Expectations? (*ICS'15*)

A. Calotoiu et al.: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes (*SC'13*)

Work supported by: DFG, SPPEXA, FNSNF, Department of Energy, United States of America

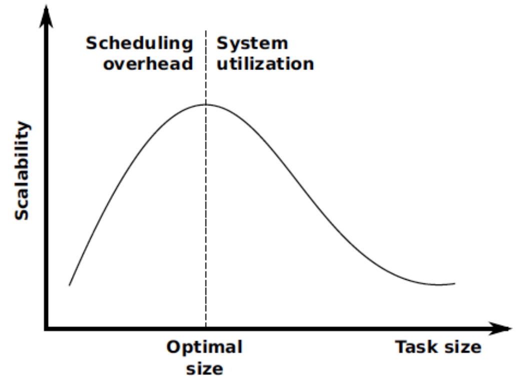
GEFÖRDERT VOM Bundesministerium für Bildung und Forschung

Future directions

- TDG construction based on OMPT (OpenMP 5.0)
  - Support for LLVM OMP runtime & other runtimes
  - Support for parallel loops
- Analysis of resource contention overhead per individual task / loop chunk
  - Gather key PAPI counters at task / chunk level

Joint work with LLNL

- Modeling other TDG metrics:
  - Relation between granularity and average parallelism: $\pi(s)$
  - Optimal granularity?
  - Maximum degree of concurrency: $d(n)$



D. Akhmetova et al., CLUSTER '15