An Empirical Methodology for Judging the Performance of Parallel Algorithms on Heterogeneous Clusters

Jackson W. Massey, Anton Menshov, and Ali E. Yilmaz

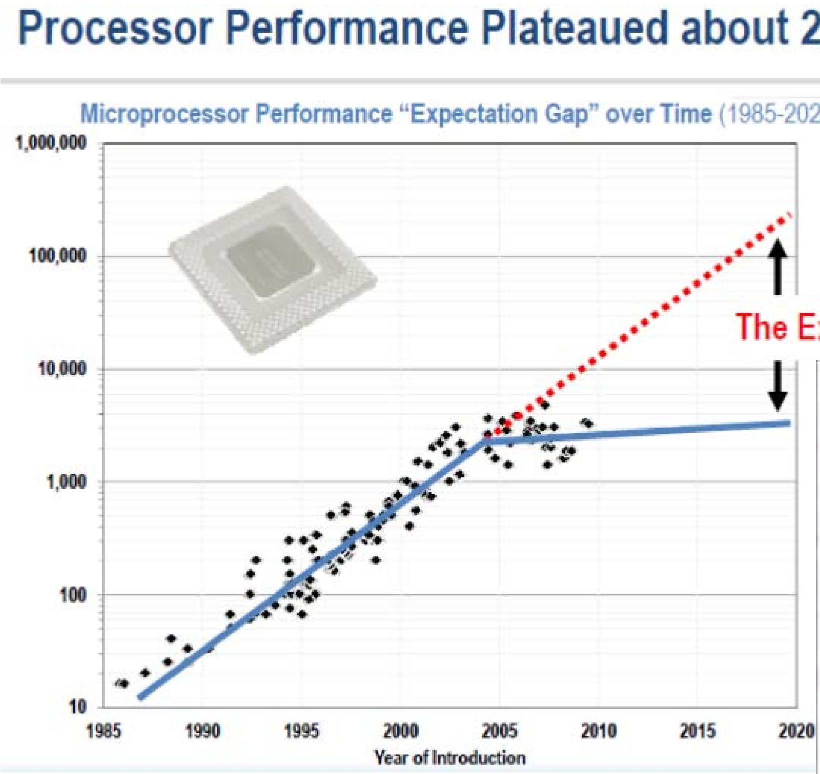
Department of Electrical & Computer Engineering
University of Texas at Austin

FEM 2016
Florence, Italy, May 2016

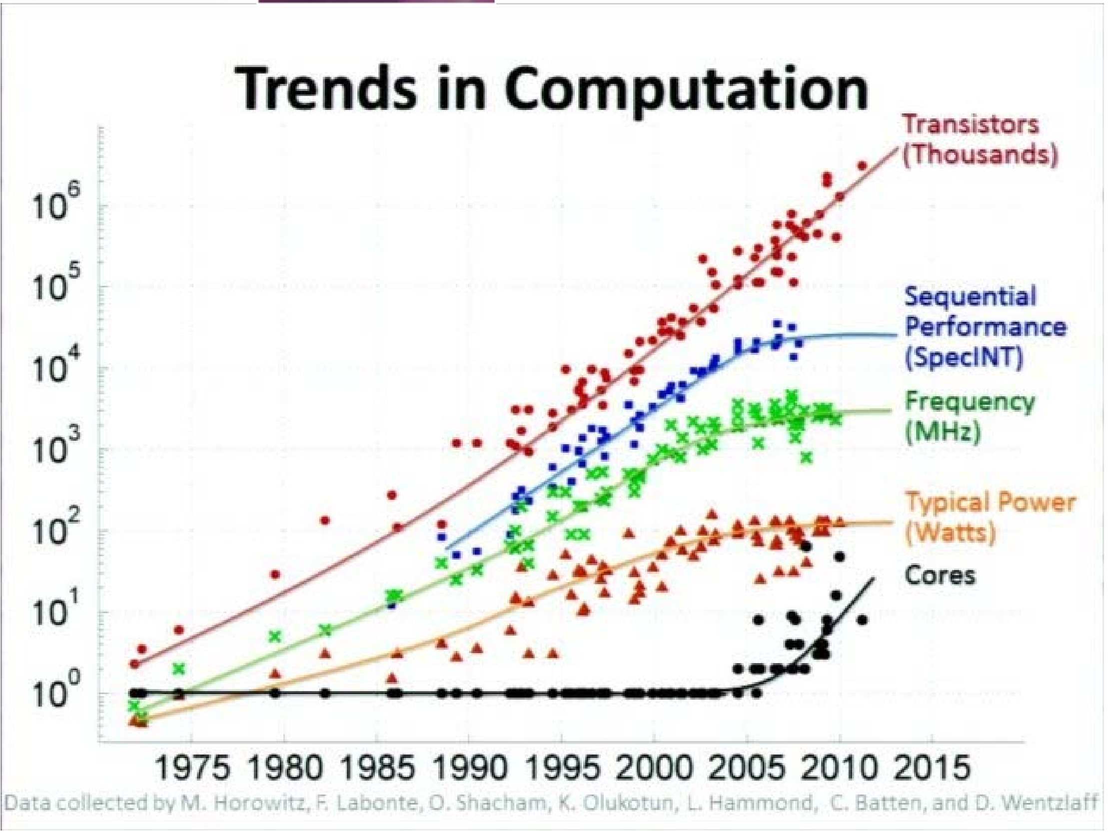
Outline

- Motivation
  - End of an Era
  - Heterogeneous Computing
  - Computational Systems
- Proposed Methodology
  - Generalized Parallel Efficiency Definition
  - Iso-Efficiency Maps
- Applications
  - MPI vs. OpenMP vs. MPI/OpenMP for Multi-core CPU with MOM
  - Multi-core CPU vs. MIC vs. Multi-core CPU+MIC with MOM
  - Iterative vs. Direct MOM Solver
- Summary & Conclusions

Limits of Sequential Computing



"The Future of Computing Performance-Game Over or Next Level?"
S. H. Fuller, L. I. Millett, Eds.; National Research Council, 2011.

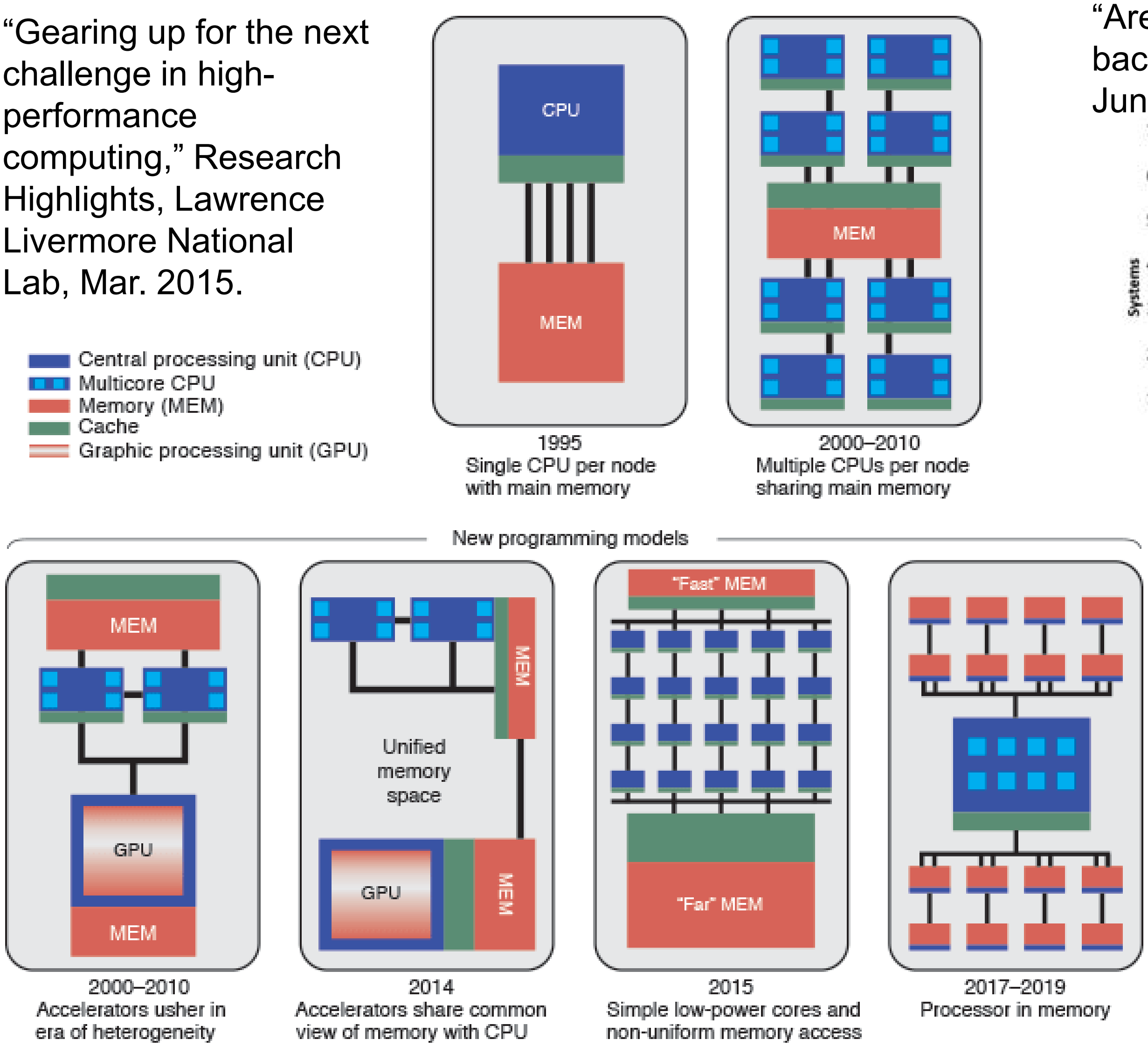


Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten, and D. Wentzlaff

"The end of the exponential runup in uniprocessor performance and the market saturation of the general-purpose processor mark the end of the "killer micro." <u>This is a golden time for innovation in computing architectures and software</u>. We have already begun to see diversity in computer designs to optimize for such metrics as power and throughput. The next generation of discoveries will require advances at both the hardware and the software levels."

Interesting Times

"Gearing up for the next challenge in high-performance computing," Research Highlights, Lawrence Livermore National Lab, Mar. 2015.



1995
Single CPU per node with main memory

2000–2010
Multiple CPUs per node sharing main memory

2000–2010
Accelerators usher in era of heterogeneity

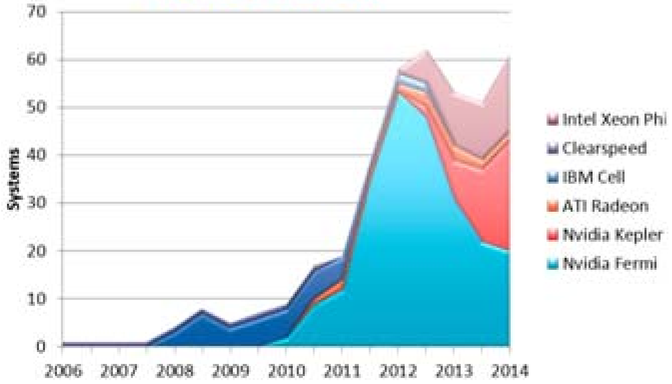
2014
Accelerators share common view of memory with CPU

2015
Simple low-power cores and non-uniform memory access

2017–2019
Processor in memory

"Are supercomputing's elite turning backs on accelerators?" hpcwire.com, June 2014

Accelerators



THE SEMICONDUCTOR INDUSTRY WILL SOON ABANDON ITS PURSUIT OF MOORE'S LAW.
NOW THINGS COULD GET A LOT MORE INTERESTING.

Nature, Feb. 2016

"The industry road map released next month will for the first time lay out a research and development plan that is not centered on Moore's law…"

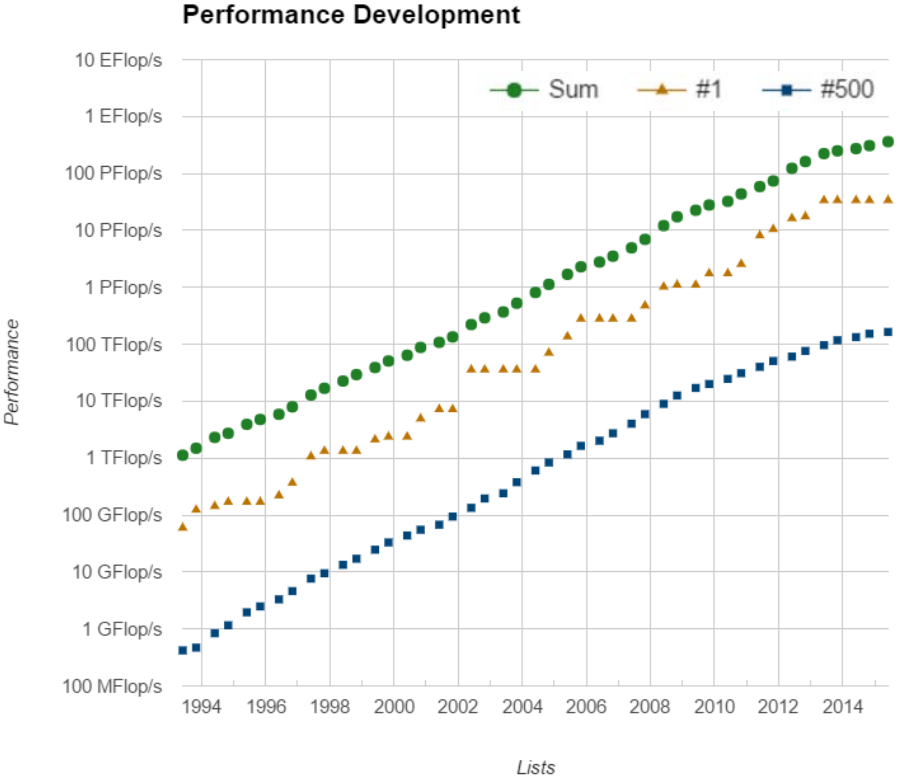
Clusters of Heterogeneous Nodes

Top Supercomputers 2015 (top500.org)

- 1. Tianhe-2
  Intel Xeon E5 + Xeon Phi 31S1P
  12 Cores 2.2 GHz
- 2. Titan - Cray XK7
  AMD Opteron 6274+ Nvidia K20x
  16 Cores 2.2 GHz
- 10. Stampede
  Intel Xeon E5 + Xeon Phi SE10P
  2x 8 Cores 2.7 GHz +
  61 Cores ~ 1070 GFLOPs

Tianhe-2

Titan

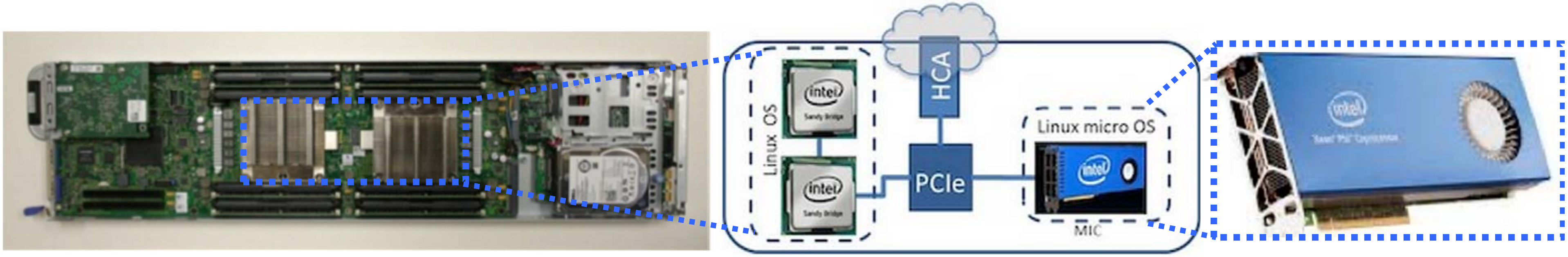


Stampede

Clusters of Heterogeneous Nodes

Stampede – Poweredge C8220 Node

2 x Intel Xeon E5-2680	Phi SE10P
- 2.7 GHz	- 1.1 GHz
- 2 x 8 cores	- 61 cores (244 threads)
- 256-bit vector unit	- 512-bit vector unit
- 0.3456 TF peak DP	- 1.074 TF peak DP



- Heterogeneous computing
  - Coordination of different types of “processors” to perform computations
    - Differences include: clock speeds, memory size/speed, instruction sets, …
  - Must re-think concepts of computational power, efficiency
  - Must account for types of processors not just number of processors

Computational Systems for Science & Engineering

- Ingredients of “computational systems” (e.g., for solving EM problems)
  - System = algorithm + software implementation + hardware architecture
  - Ongoing advances in each ingredient
    - Often focus on one and make abstractions (sweeping generalizations/ simplifications) about others, assuming/hoping

$$\overset{\text{“best”}}{\text{system}} = \overset{\text{“best”}}{\text{algorithm}} \cap \overset{\text{“best”}}{\text{implementation}} \cap \overset{\text{“best”}}{\text{hardware}}$$

      and improved ingredient => improved system
    - Enabled tremendous progress, becoming more difficult/less valid: Algorithm dependent hardware performance (architects often recognize this), implementation dependent algorithm performance (coders often recognize this) …
  - No “universal best system” for all problems but some systems (much) better for important problem classes
  - $\underset{\text{this work}}{\underline{\underline{\text{How to judge different systems?}}}}$ Define problem, $\underline{\underline{\text{define metrics}}}$, apply system, collect data, observe/explain/compare, …

Computational Systems for Science & Engineering

- Metrics/figures of merit/performance measures for judging CEM systems
  - Most important ones:
    1. Accuracy: Is error in solution acceptable? (Need a reference)
    2. Cost: Is problem solved fast enough? (Need a lower limit)
    3. Efficiency: How much of available computational power is wasted? (Must define available computational power)
    4. Scalability: How much should system grow when problem grows to keep metrics acceptable? (Must define paths to grow problem, system)
    5. … stability/robustness, error convergence rate, portability, user interface, …
- This work:
  - Efficiency (and scalability) on heterogeneous computers & clusters of them
  - Key concepts (computational power, workload)
  - Proposed methodology (iso-efficiency contours and acceptable performance)
  - Examples comparing different systems

Proposed Methodology

- Generalized Parallel Efficiency Definition
- Iso-Efficiency Maps

Efficiency for Heterogeneous Clusters

- System of interest = algorithm + software implementation + P processors

- Well-known for homogeneous clusters of P identical processors

W: workload

$t_p(W)$: wallclock time to solve problem using only processor p

$t_{\text{obs}}(W)$: wallclock time to solve problem using all P processors

$$e(P,W) \triangleq \frac{t_1(W)}{P t_{\text{obs}}(P,W)}$$: (parallel) efficiency of system

- Generalization to heterogeneous clusters of different types of processors

L. Pastor and J. L. B. Orero, "An efficiency and scalability model for heterogeneous clusters," in *Proc. IEEE Conf. Cluster Comp.*, Oct. 2001, pp. 427-434.

$$e(C_{\text{tot}},W) \triangleq \frac{1/t_{\text{obs}}(P,W)}{C_{\text{tot}}(P,W)}$$: (parallel) efficiency of system

$$C_{\text{tot}}(P,W) \triangleq \sum_{p=1}^{P} C_p(W)$$: total comp. power available to system

$$C_p(W) \triangleq \frac{1}{t_p(W)}$$: average comp. power of system using only processor p

Efficiency for Heterogeneous Clusters

- System of interest = algorithm + software implementation + P processors

L. Pastor and J. L. B. Orero, “An efficiency and scalability model for heterogeneous clusters,” in *Proc. IEEE Conf. Cluster Comp.*, Oct. 2001, pp. 427-434.

$$e(C_{\text{tot}}, W) \triangleq \frac{1/t_{\text{obs}}(P,W)}{C_{\text{tot}}(P,W)}$$: (parallel) efficiency of system

$$C_{\text{tot}}(P,W) \triangleq \sum_{p=1}^{P} C_p(W)$$: total comp. power available to system

$$C_p(W) \triangleq \frac{1}{t_p(W)}$$: average comp. power of system using only processor p

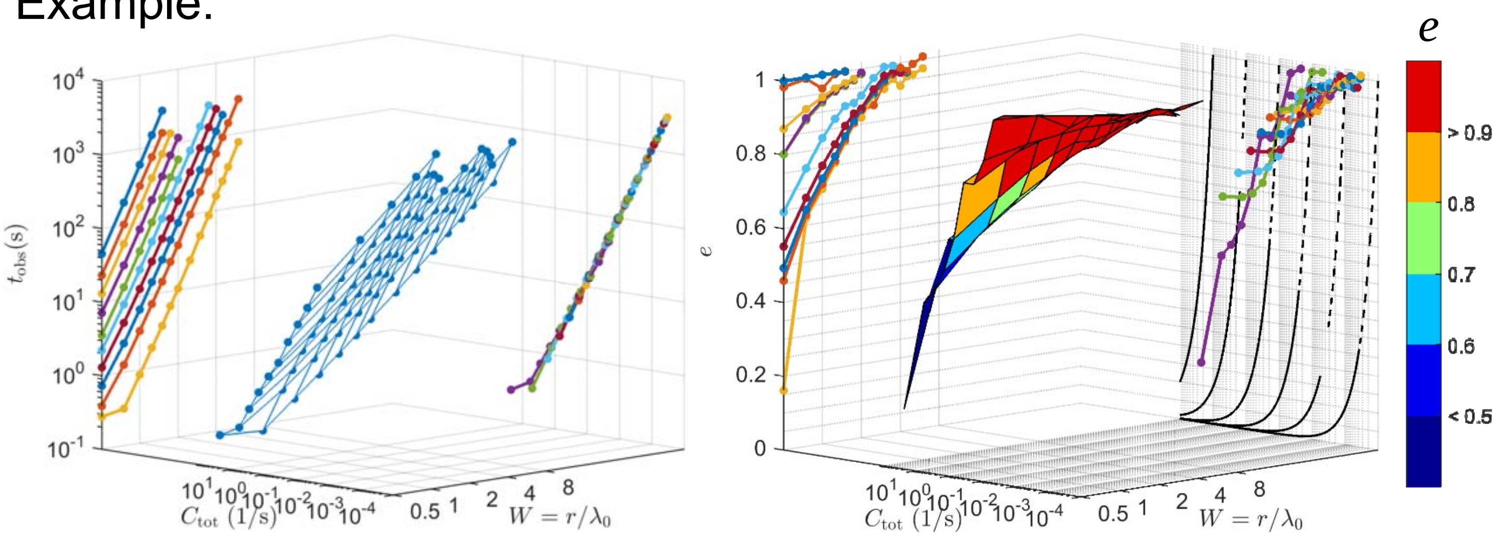
- Properties & interpretation
  - Salient feature: Define W to be independent of system!
  - C_{tot}: Part of work that could have been done per sec (if system efficiency=1)
  - $1/t_{\text{obs}}$: Part of work actually done per sec
  - Reduces to usual definition for homogeneous clusters
  - C_{tot}, e sensitive to algorithm, software implementation, number/type of processors used & workload => Can study effect of each ingredient

Iso-Efficiency Maps

- System of interest = algorithm + software implementation + P processors

$$\boxed{e(C_{\text{tot}},W) \triangleq \frac{1/t_{\text{obs}}(P,W)}{C_{\text{tot}}(P,W)} \text{ : (parallel) efficiency of system}}$$

- Maps of iso-efficiency contours
  - Generate by sweeping P, W and recording t_{obs}. Plot in $C_{\text{tot}} - W$ plane.
  - Example:



C_{tot}: Part of work that could have been done per sec

$1/t_{\text{obs}}$: Part of work actually performed per sec

Pitfall: Must find a way to estimate $t_p(W)$ for large W. Extrapolating from larger C_{tot} often too rosy. Extrapolating from smaller W may not be possible.

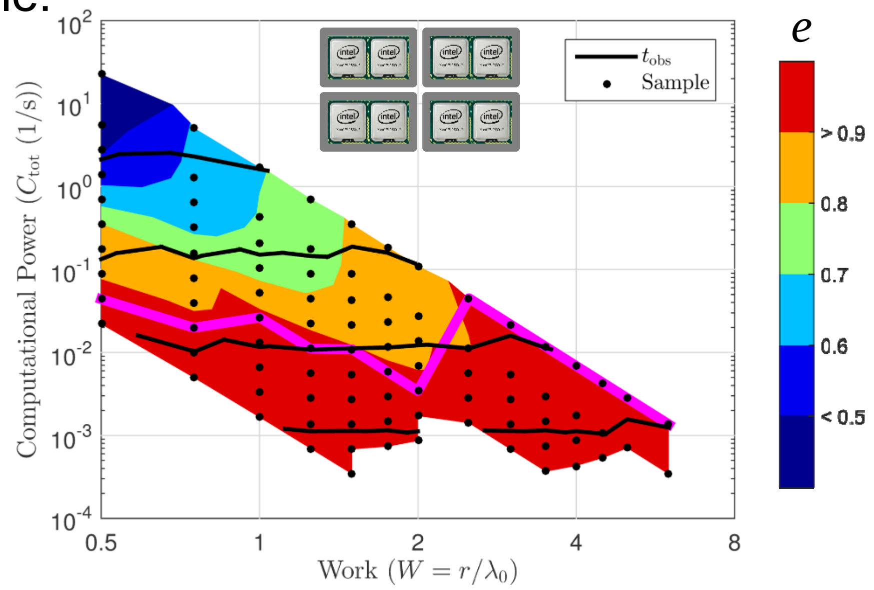
F. Wei and A. E. Yılmaz, "A Systematic Approach to Judging Parallel Algorithms: Acceptable Parallelization Regions in the N-P Plane," in *Proc. FEM '14*, May 2014.

Iso-Efficiency Maps

- System of interest = algorithm + software implementation + P processors

$$e(C_{\text{tot}}, W) \triangleq \frac{1/t_{\text{obs}}(P,W)}{C_{\text{tot}}(P,W)}$$: (parallel) efficiency of system

- Maps of iso-efficiency contours
  - Generate by sweeping P, W and recording t_{obs}. Plot in $C_{\text{tot}} - W$ plane.
  - Example:



- Specify requirements: e.g.,
  - acceptable efficiency $e \geq 0.9$
  - must do >0.1% of work per sec
- Find highest C_{tot} that meets requirements
- Pitfall: Must find a way to estimate reference $t_p(W)$ for large W. Extrapolating from larger C_{tot} often too rosy. Extrapolating from smaller W may not be possible.

(F. Wei and A. E. Yılmaz, "A Systematic Approach to Judging Parallel Algorithms: Acceptable Parallelization Regions in the N-P Plane," in *Proc. FEM '14*, May 14.)

C_{tot}: Part of work that could have been done per sec

$1/t_{\text{obs}}$: Part of work actually done per sec

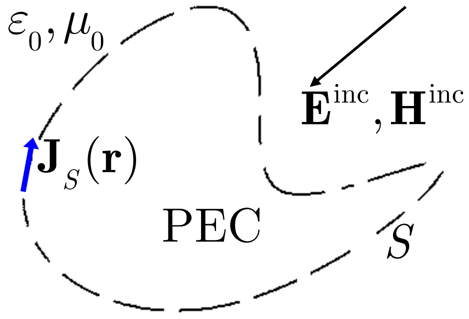
Applications

- **Benchmark Description**
- System Evaluation for Algorithm I – Iterative Solver
- System Evaluation for Algorithm II –Direct Solver
- Computational System Comparison

Example Algorithms

- CFIE for perfectly conducting closed surface *S*

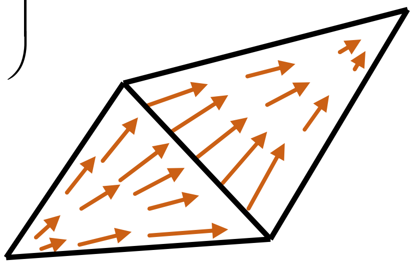
$$\text{EFIE}:\mathbf{E}^{\mathrm{inc}}(\mathbf{r})\Big|_{\tan S} = \left. \begin{matrix} j\omega\mu_0 \iint_S \mathbf{J}_S(\mathbf{r}')g(\mathbf{R})dS' \\ -\dfrac{\nabla}{j\omega\varepsilon_0}\iint_S \nabla'\bullet\mathbf{J}_S(\mathbf{r}')g(\mathbf{R})dS' \end{matrix}\right|_{\tan S}$$



$$\text{MFIE}:\hat{\mathbf{n}}\times\mathbf{H}^{\mathrm{inc}}(\mathbf{r}) = \mathbf{J}_{kl}^{\mathrm{S}} - \hat{\mathbf{n}}\times\left(\nabla\times\iint_S \mathbf{J}_S(\mathbf{r}')g(\mathbf{R})dS'\right)$$

$$\text{CFIE} = \alpha\text{EFIE} + (1-\alpha)\eta_0\text{MFIE}$$

$$g(\mathbf{R}) = e^{-jk_0R}\Big/4\pi R; \qquad \eta_0 = \sqrt{\mu_0/\varepsilon_0}$$



Rao, Wilton, Glisson, *IEEE Trans. AP*, May 1982.

- Method of moments solver

$$\mathbf{J}_S(\mathbf{r}) \cong \sum_{n=1}^{N}\mathbf{I}[n]\mathbf{f}_n(\mathbf{r}) \Rightarrow \mathbf{Z}_{N\times N}\mathbf{I}_{N\times N_{\mathrm{RHS}}} = \mathbf{V}_{N\times N_{\mathrm{RHS}}}$$

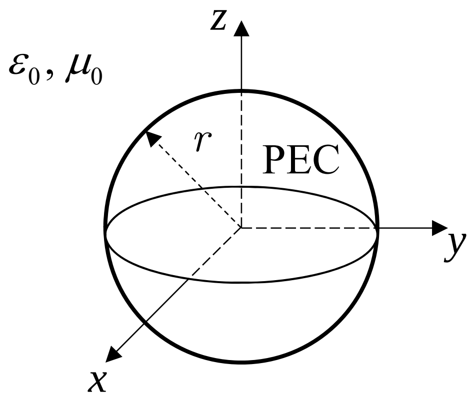
- Computational complexity

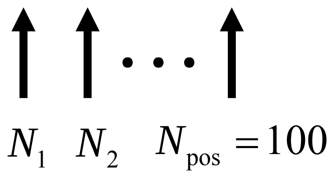
Matrix fill: $O(N^2)$ Algorithm I=> Iterative solve (TFQMR): $O(N_{\mathrm{iter}}N_{\mathrm{RHS}}N^2)$

Algorithm II=> Direct solve (MKL LAPACK + ScaLAPACK*): $O(N^3 + N_{\mathrm{RHS}}N^2)$

*ScaLAPACK block size = 512

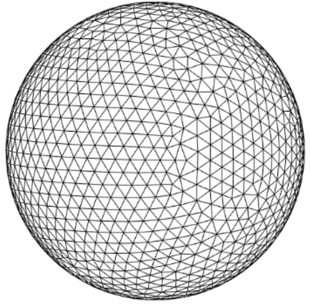
Sample Workloads





Hertzian dipole

$r = \lambda_0,\ N = 4314$



r/λ_0	N
0.5	1071
0.75	2421
1	4314
1.25	6741
1.5	9693
1.75	13 269
2	17 307
2.5	27 120
3	38 853
3.5	53 085
4	69 192
5	107 949
6	155 310
7	211 947

- Asymptotic algorithm costs:

$$t_{\text{fill}} \propto \left(\frac{r}{\lambda_0}\right)^4$$

$$\text{Algorithm I } t_{\text{solve}} \propto \left(\frac{r}{\lambda_0}\right)^4 N_{\text{pos}} N_{\text{iter}}$$

$$\text{Algorithm II } t_{\text{solve}} \propto \left(\frac{r}{\lambda_0}\right)^6 + \left(\frac{r}{\lambda_0}\right)^4 N_{\text{pos}}$$

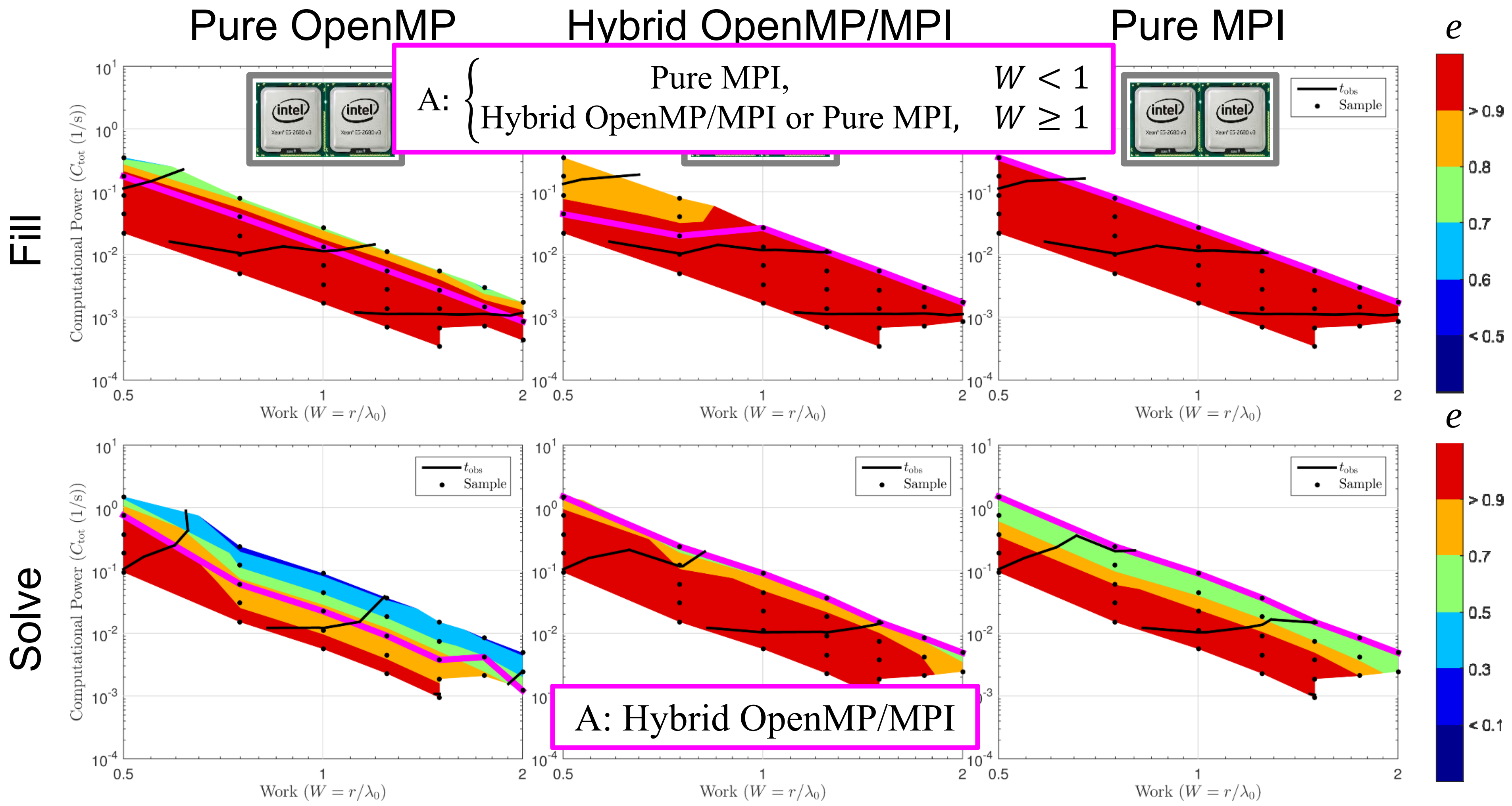
- Workload definition: $W = \frac{r}{\lambda_0}$
- Fill acceptable efficiency $e \geq 0.9$
- Solve acceptable efficiency $e \geq 0.5$
- Must do >0.1% of work per sec
- Find reference $t_p(W)$ for large W by extrapolating from $t_p(W)$ for small W using asymptotic expressions

Applications

- Benchmark Description
- System Evaluation for Algorithm I – Iterative Solver
- System Evaluation for Algorithm II –Direct Solver
- Computational System Comparison

Iterative Solver
Intranode CPU Study

Q: Overall, which parallel implementation is best?



$$\text{A: } \begin{cases} \text{Pure MPI,} & W < 1 \\ \text{Hybrid OpenMP/MPI or Pure MPI,} & W \geq 1 \end{cases}$$

A: Hybrid OpenMP/MPI

1 MPI process
Vary OpenMP threads (1-16)

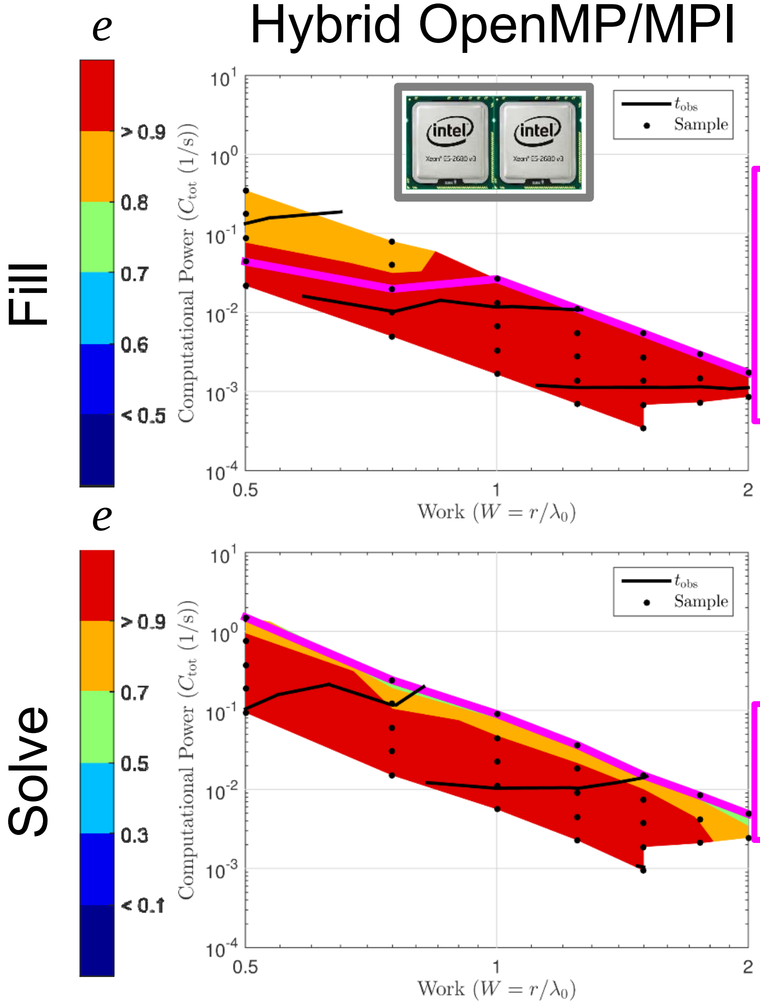
Vary MPI processes (1-2)
Vary OpenMP threads (1-8)

Vary MPI processes (1-16)
1 OpenMP thread

Q: Which process/thread configuration is best?

Iterative Solver Intranode CPU Study

Hybrid OpenMP/MPI



$$\text{A:} \begin{cases} \text{2 MPI processes with 1–2 OpenMP threads,} & W < 1 \\ \text{2 MPI processes with 8 OpenMP threads,} & W \geq 1 \end{cases}$$

A: 2 MPI processes with 8 OpenMP threads

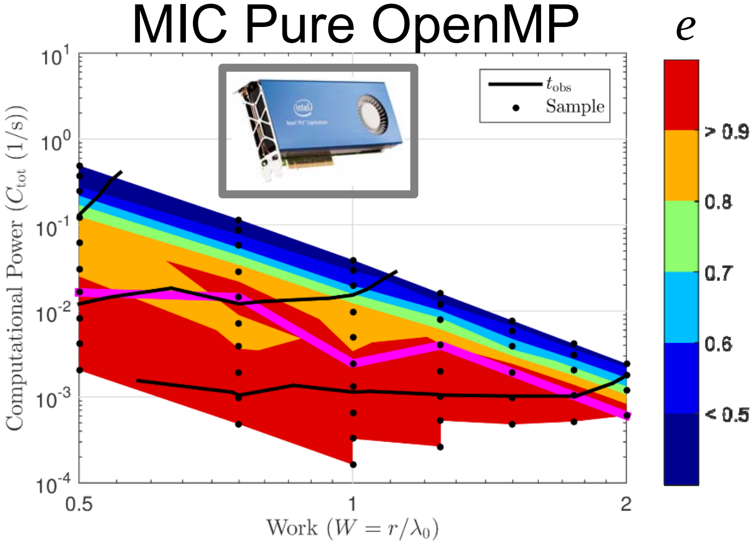
Vary MPI processes (1-2)
Vary OpenMP threads (1-8)

Q: Which process/thread configuration is best?

Iterative Solver Intranode MIC Study

MIC Pure OpenMP

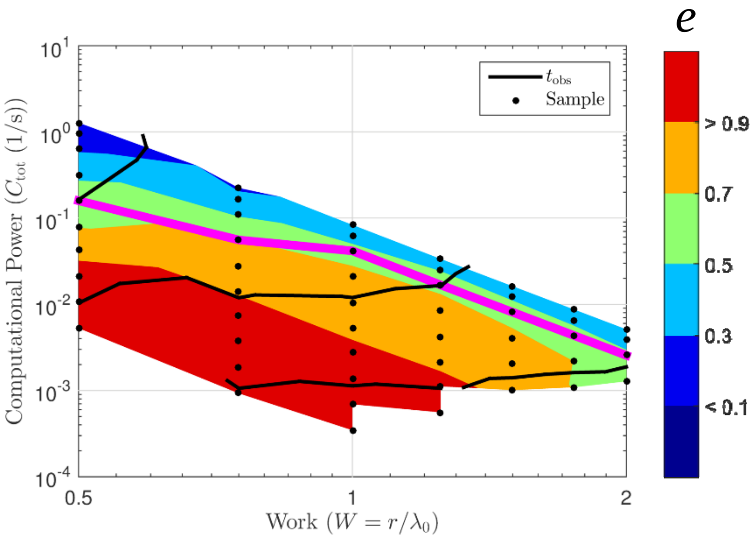
Fill



- MIC Pure OpenMP
  - 1 MPI process
  - Vary OpenMP threads (1-240)

$$\text{A:} \begin{cases} \text{15–30 OpenMP threads,} & W < 1.25 \\ \text{60 OpenMP threads,} & W \geq 1.25 \end{cases}$$

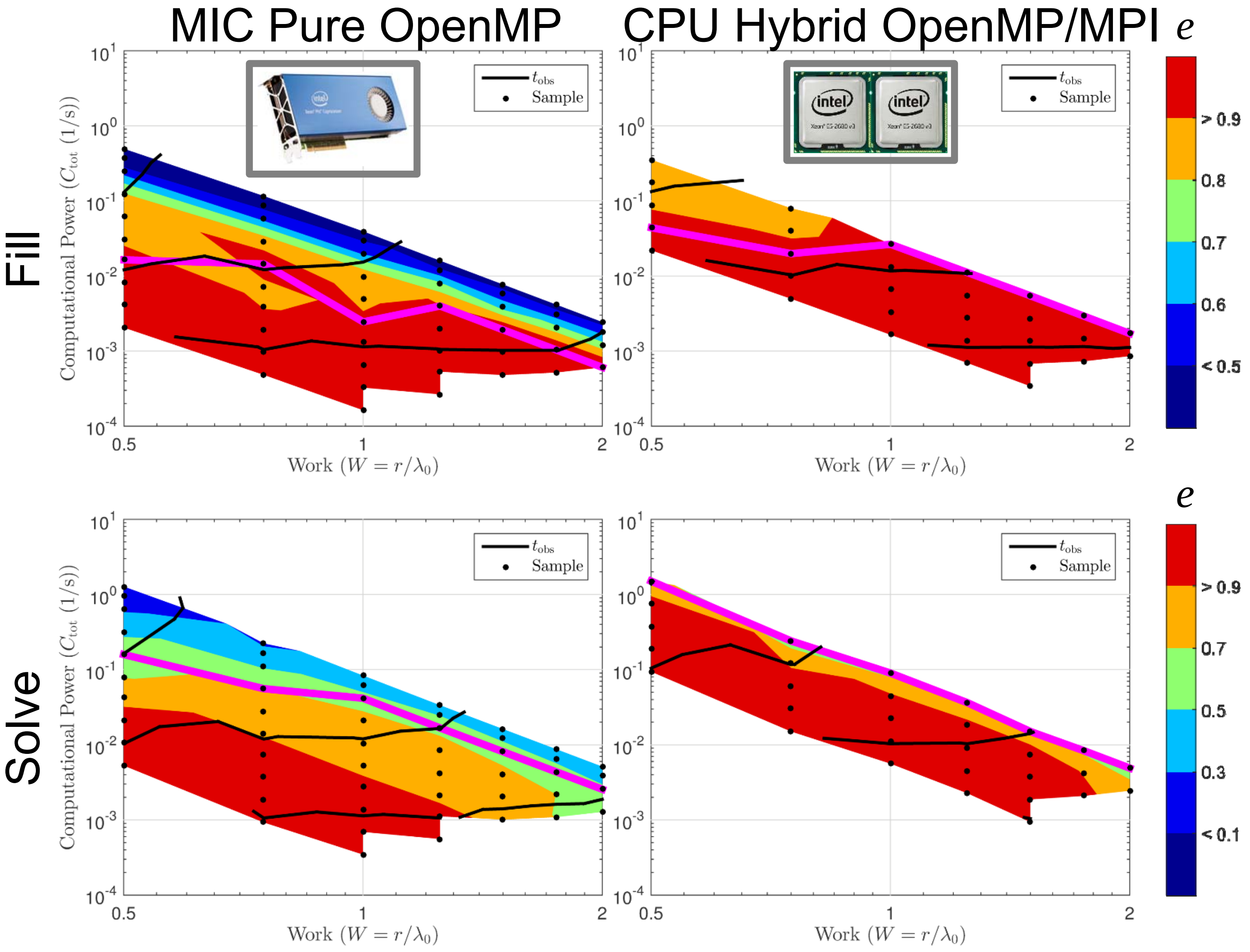
Solve



$$\text{A:} \begin{cases} \text{60 OpenMP threads,} & W < 1 \\ \text{120 OpenMP threads,} & W \geq 1 \end{cases}$$

Iterative Solver
Intranode MIC/CPU Study

Q: Which hardware + parallel implementation is best?



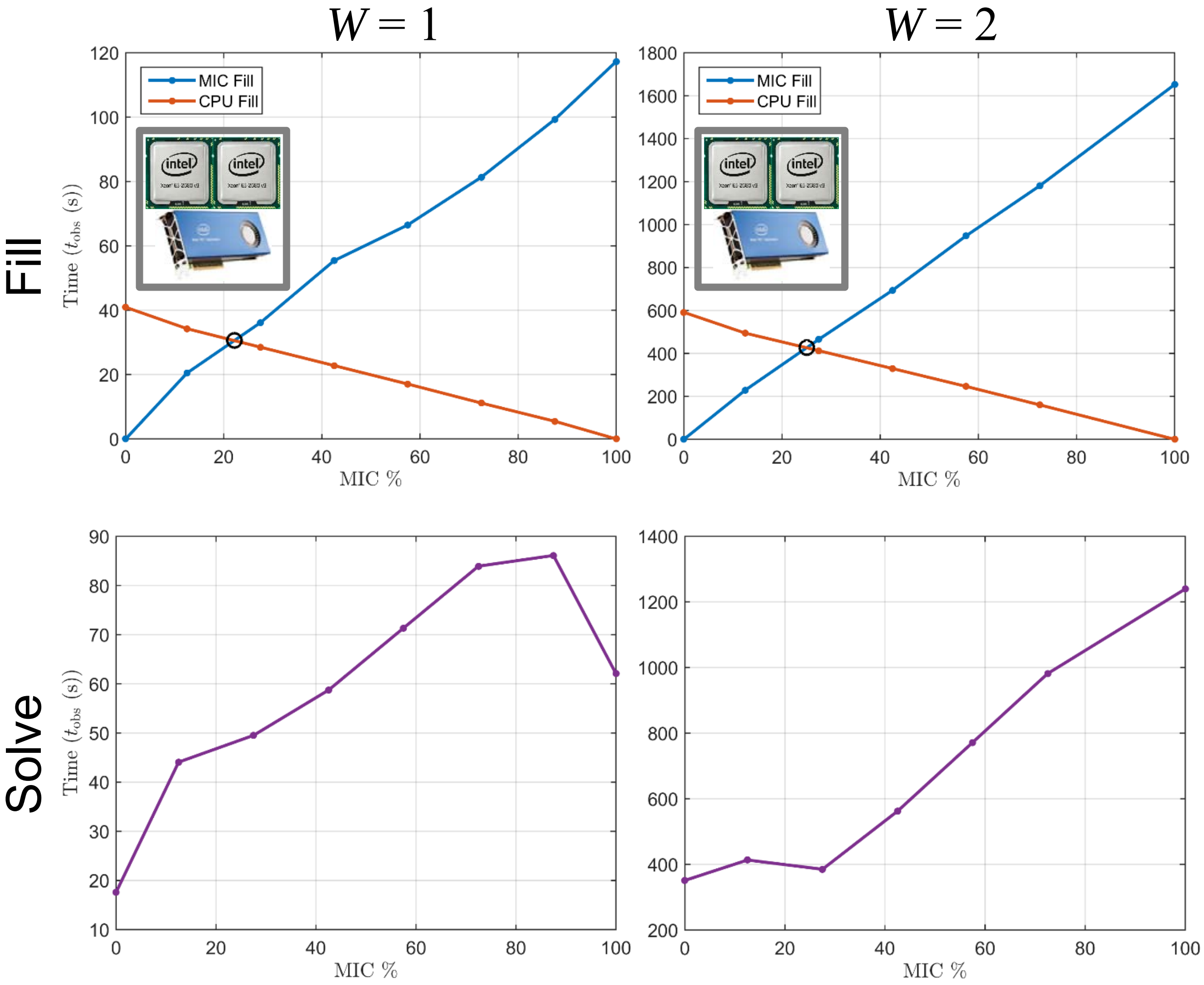
- MIC Pure OpenMP
  - 1 MPI process
  - Vary OpenMP threads (1-240)

A: CPU Hybrid OpenMP/MPI

- CPU Hybrid OpenMP/MPI
  - Vary MPI processes (1-2)
  - Vary OpenMP threads (1-8)

A: CPU Hybrid OpenMP/MPI

Iterative Solver Intranode CPU+MIC Study



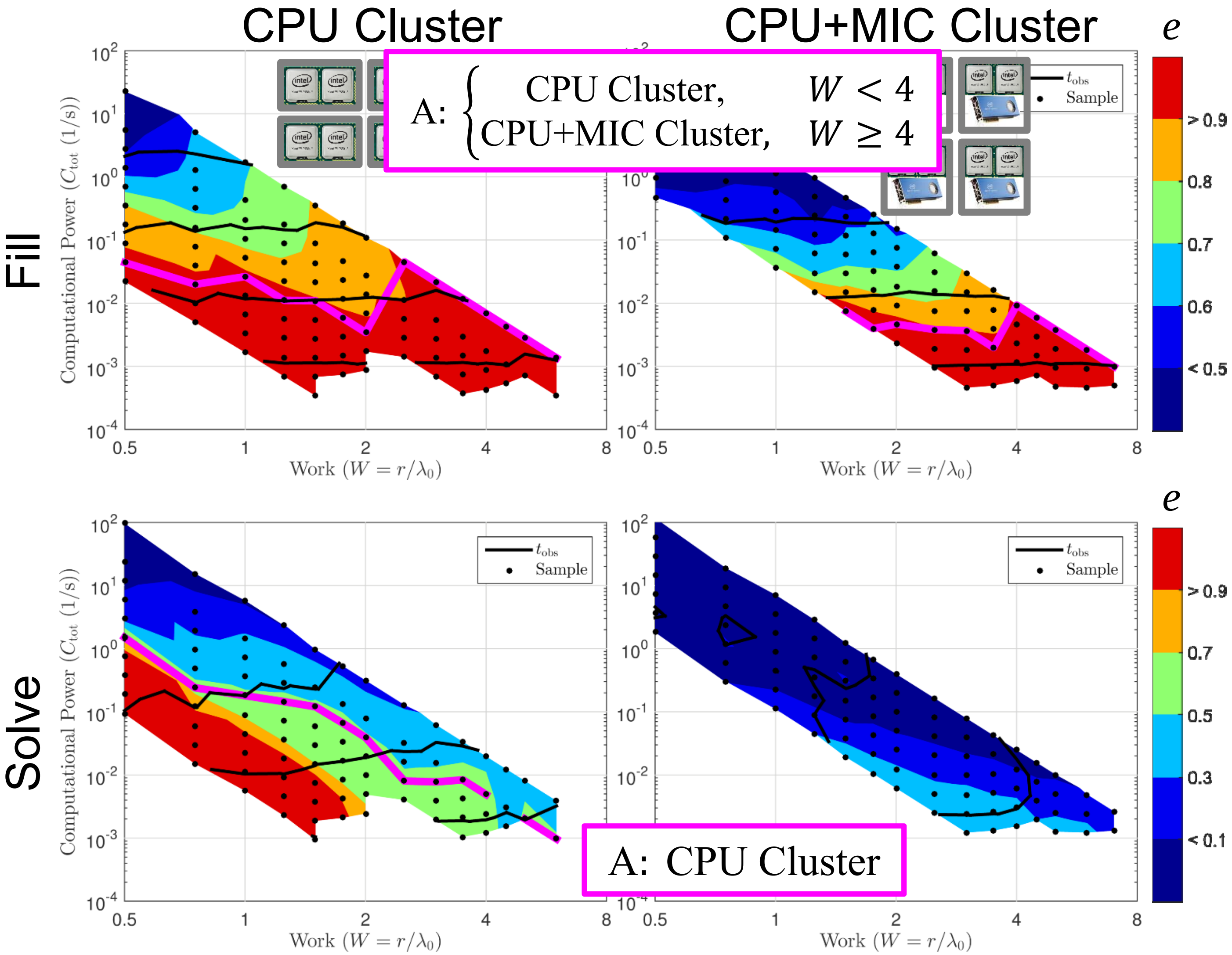
- Symmetric MIC run*
- CPU
  - 2 MPI processes
  - 8 OpenMP threads each
- MIC
  - 1 MPI process
  - 60 OpenMP threads
- Find optimal workload balance

Use 25% workload on MIC

* Simplest method to use MIC with CPU (not ideal)

Iterative Solver
Internode CPU+MIC Study

Q: Which hardware + parallel implementation is best?



$$\text{A: } \begin{cases} \text{CPU Cluster,} & W < 4 \\ \text{CPU+MIC Cluster,} & W \geq 4 \end{cases}$$

A: CPU Cluster

$$C_{\text{tot}} = P_{\text{nodes}} 16 C_{\text{CPUcore}}$$

$$C_{\text{tot}} = P_{\text{nodes}} [16 C_{\text{CPUcore}} + 60 C_{\text{MICcore}}]$$

- Varying number of nodes (<1-64)
- CPU / node
  - 2 MPI processes
  - 8 OpenMP threads each
- MIC / node
  - Symmetric MIC run*
  - 1 MPI process
  - 60 OpenMP threads
  - 25% workload

* Simplest method to use MIC with CPU (not ideal)

Applications

- Benchmark Description
- System Evaluation for Algorithm I – Iterative Solver
- System Evaluation for Algorithm II –Direct Solver
- Computational System Comparison

Direct Solver
Intranode CPU Study

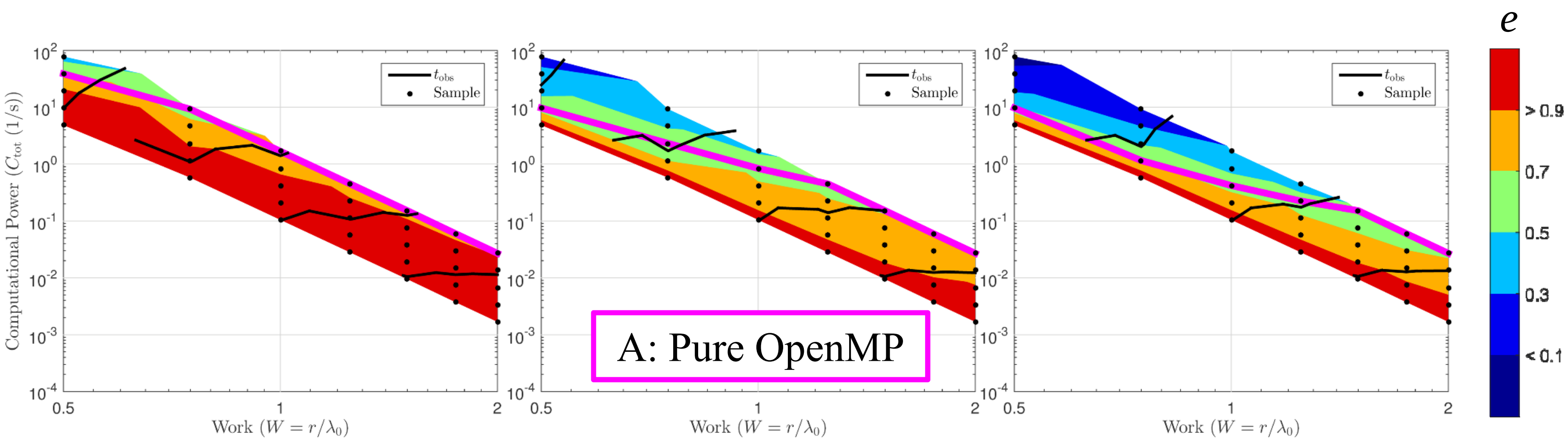
Q: Overall, which parallel implementation is best?

Pure OpenMP

Hybrid OpenMP/MPI

Pure MPI

Solve



A: Pure OpenMP

1 MPI process
Vary OpenMP threads (1-16)

Vary MPI processes (1-2)
Vary OpenMP threads (1-8)

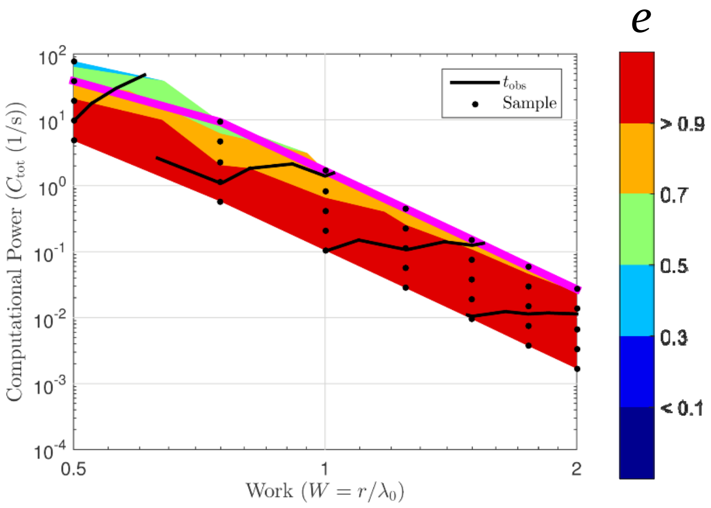
Vary MPI processes (1-16)
1 OpenMP thread

Direct Solver Intranode CPU Study

Q: Which process/thread configuration is best?

Pure OpenMP

Solve



1 MPI process
Vary OpenMP threads (1-16)

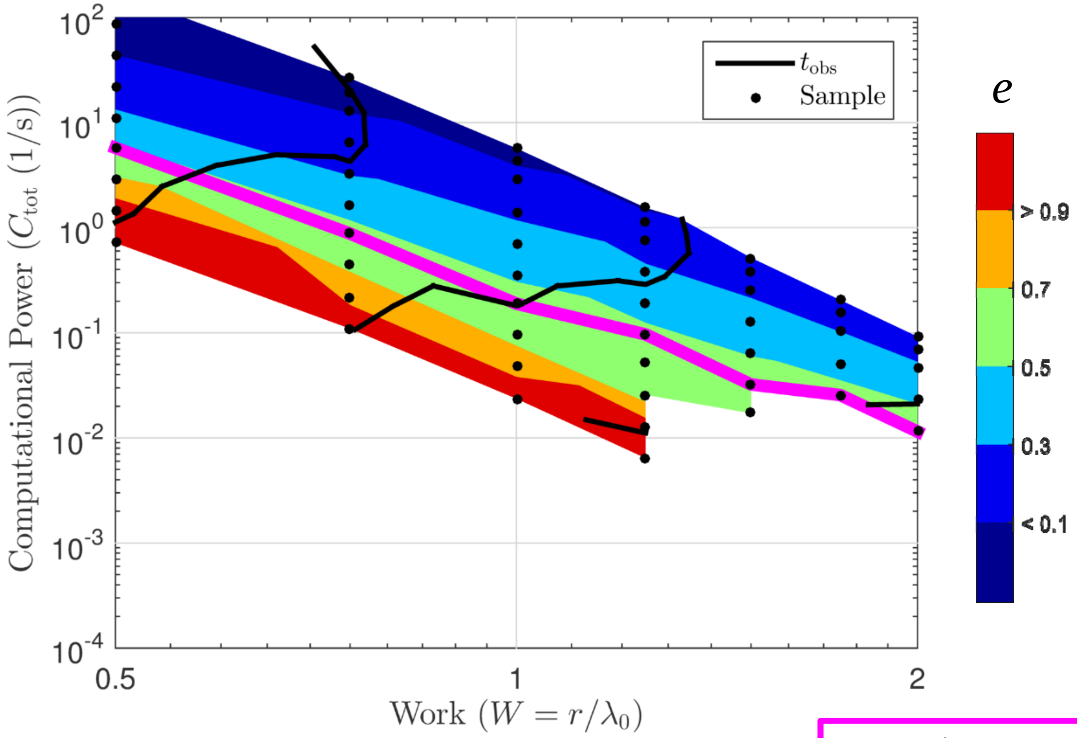
$$\text{A: } \begin{cases} \text{1 MPI process with 8 OpenMP threads,} & W < 0.75 \\ \text{1 MPI process with 16 OpenMP threads,} & W \geq 0.75 \end{cases}$$

Q: Which process/thread configuration is best?

Direct Solver
Intranode MIC Study

MIC Pure OpenMP

Solve

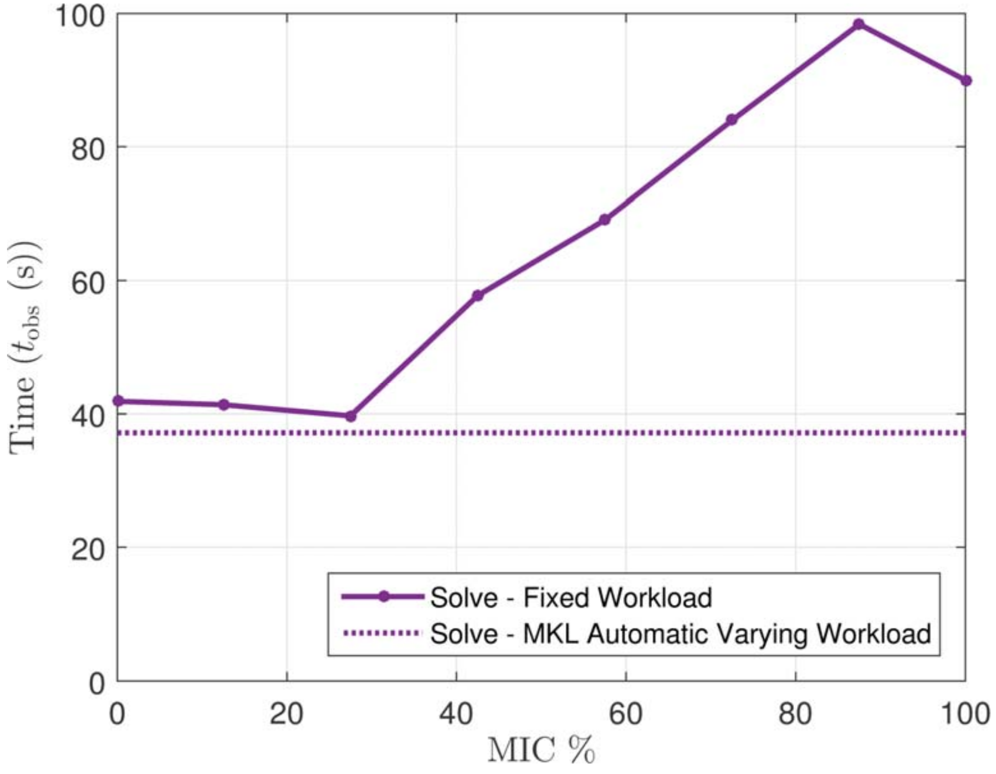


- MIC Pure OpenMP
  - 1 MPI process
  - Varying OpenMP threads (1-240)

$$A: \begin{cases} 8 \text{ OpenMP threads}, & W < 1.25 \\ 15 \text{ OpenMP threads}, & 1.25 \le W < 1.75 \\ 30 \text{ OpenMP threads}, & W \ge 1.75 \end{cases}$$

Direct Solver
Intranode CPU+MIC Study

Solve



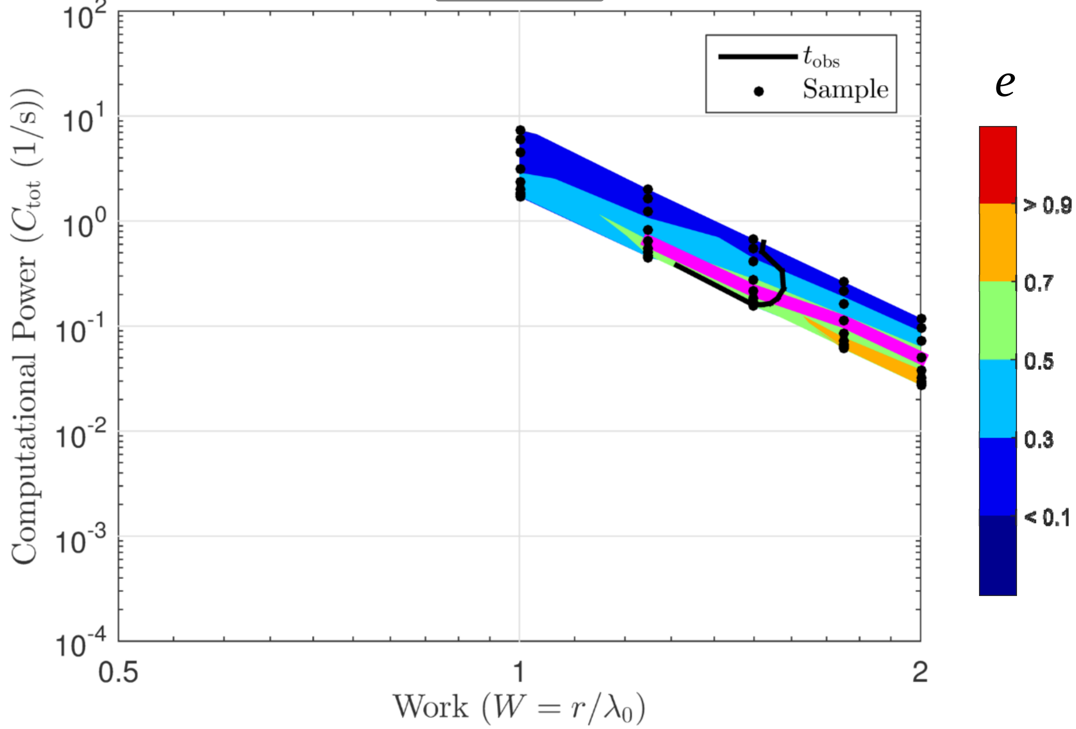
- $W = 2$
- Direct solve
  - Automatic offload (Intel MKL)
  - CPU: 1 MPI process with 16 OpenMP threads
  - MIC: 30 OpenMP threads

Direct: Use MKL automatic varying workload

Direct Solver
Intranode CPU+MIC Study

Q: Which process/thread configuration is best?

Solve



- CPU+MIC
  - CPU: 1 MPI process with 16 OpenMP threads
  - MIC: Varying OpenMP threads (1-240), $T_{\text{MICthreads}}$
  - Automatic offload: MKL automatic varying workload

$$C_{\text{tot}} = 16 C_{\text{CPUcore}} + T_{\text{MICthreads}} C_{\text{MICcore}}$$

$$\text{A:} \begin{cases} 30 \text{ OpenMP threads}, & 1.25 \leq W < 1.75 \\ 60 \text{ OpenMP threads}, & W \geq 1.75 \end{cases}$$

Direct Solver Intranode Study

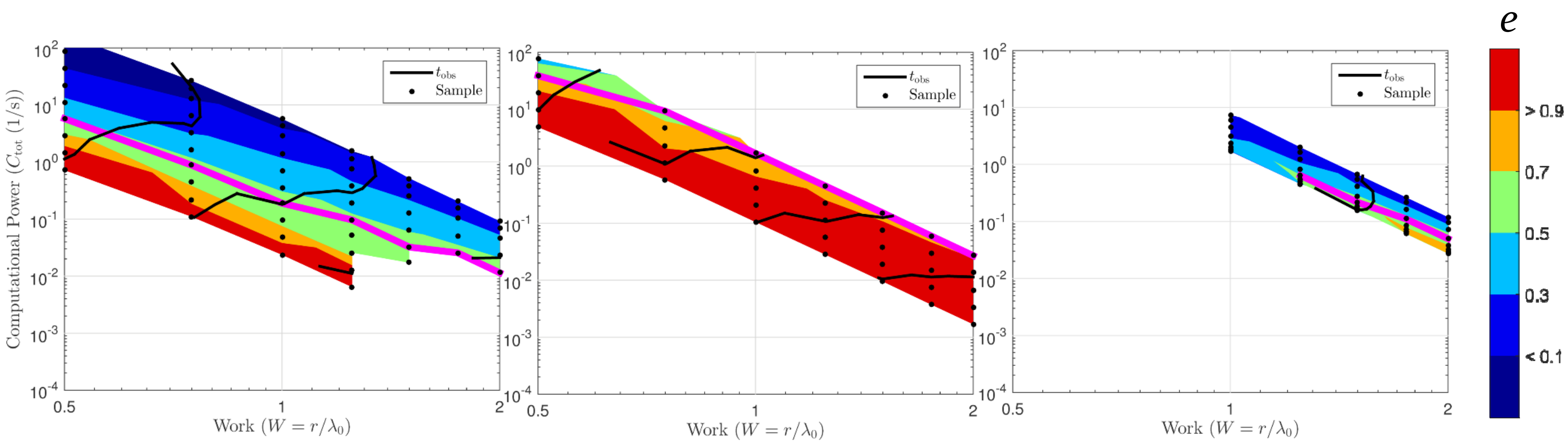
Q: Which hardware + parallel implementation is best?

MIC Pure OpenMP

CPU Pure OpenMP

CPU+MIC

Solve



MIC: 1 MPI process with varying OpenMP threads (1-240)

CPU: 1 MPI process with varying OpenMP threads (1-16)

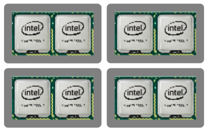
CPU: 1 MPI process with 16 OpenMP threads
MIC: Varying OpenMP threads (1-240)
MKL automatic offloading

$$\text{A:} \begin{cases} \text{CPU Pure OpenMP,} & W < 1.25 \\ \text{CPU+MIC,} & W \geq 1.25 \end{cases}$$

Direct Solver
Internode CPU+MIC Study

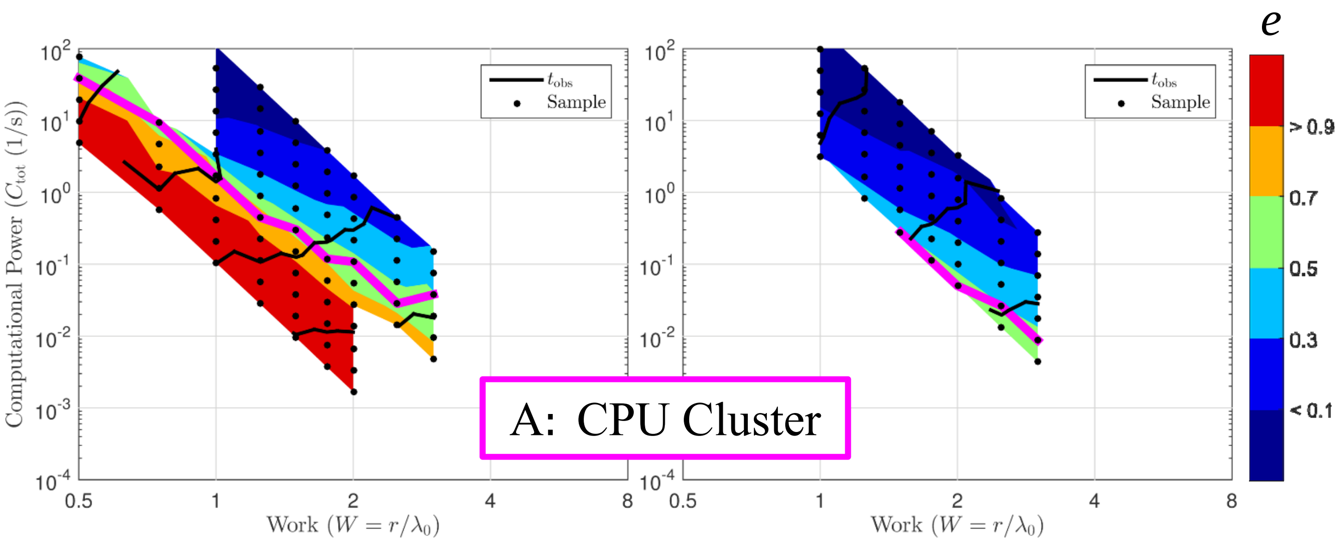
Q: Which hardware + parallel implementation is best?

CPU Cluster



CPU+MIC Cluster

Solve



A: CPU Cluster

$$C_{\text{tot}} = P_{\text{nodes}} 16 C_{\text{CPUcore}}$$

$$C_{\text{tot}} = P_{\text{nodes}} [16 C_{\text{CPUcore}} + 60 C_{\text{MICcore}}]$$

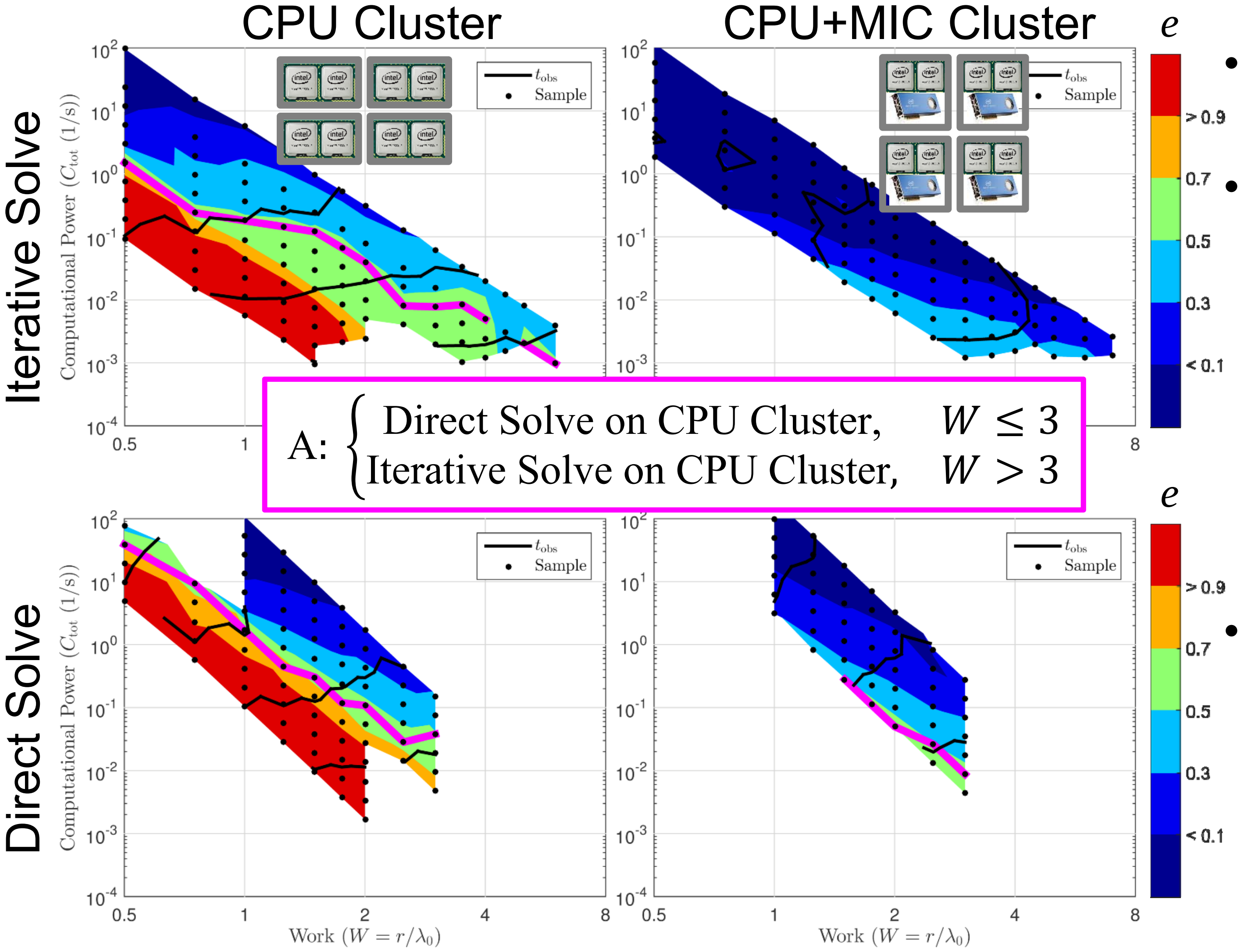
- Varying nodes (<1-64), P_{nodes}
- CPU / node
  - 1 MPI process
  - 16 OpenMP threads each
- MIC / node
  - MKL automatic offloading
  - 60 OpenMP threads
  - MKL automatic varying workload

Applications

- Benchmark Description
- System Evaluation for Algorithm I – Iterative Solver
- System Evaluation for Algorithm II –Direct Solver
- **Computational System Comparison**

Direct vs. Iterative Solver Internode CPU+MIC Study

Q: Which computational system is better?



$$\text{A:} \begin{cases} \text{Direct Solve on CPU Cluster,} & W \leq 3 \\ \text{Iterative Solve on CPU Cluster,} & W > 3 \end{cases}$$

$$C_{\text{tot}} = P_{\text{nodes}} 16 C_{\text{CPUcore}}$$

$$C_{\text{tot}} = P_{\text{nodes}} [16 C_{\text{CPUcore}} + 60 C_{\text{MICcore}}]$$

- Varying nodes (<1-64), P_{nodes}
- CPU / node
  - Iterative: 1-2 MPI processes with up to 8 OpenMP threads
  - Direct: 1 MPI process with up to 16 OpenMP threads
- MIC / node
  - 60 OpenMP threads
  - Iterative: 1 MPI process & 25% workload
  - Direct: MKL automatic offload& varying workload

Summary & Conclusions

Observations

- Judging algorithms, software, hardware
  - Era of independently judging algorithms, implementations, and hardware is (probably) ending
  - Will not be able to (credibly) claim
    - processor p1 is better/faster/more energy efficient/… than processor p2 without mentioning algorithm & software properties
    - algorithm A is better/faster/… than algorithm B without mentioning software & hardware properties
  - Must judge entire system (algorithm + software implementation + hardware)
    - can still judge ingredients but in context
    - faster not always better, must evaluate cost!

      => Q: Which one is better? Destination 100km away:

      (a) Drive in 1h or 2h? A: Of course faster is better.

      (b) Drive in 1h spending 10L of fuel or 2h spending 1L of fuel? A: It depends…
    - (parallel) efficiency is a reasonable metric for judging cost of computational systems, even for heterogeneous computing

Empirical Approach

- Proposed methodology
  - Carefully define problem of interest
  - Define workload independent of system (not in terms of basic operations [flops])
  - Determine average computational power of system under different configurations
    - Evaluate efficiency as a function of available computational power, workload, determine iso-efficiency contours
  - Compare & contrast
- Pros & cons
  - \+ Can compare entire computational systems
  - \+ Can compare ingredients (hardware, software implementations, algorithms) by modifying only one and keeping other ingredients fixed
  - \- Requires (access to) whole system
  - \- Must generate lots of data including those from relatively inefficient simulations