# MPI+X: Opportunities and Limitations for Heterogeneous Systems

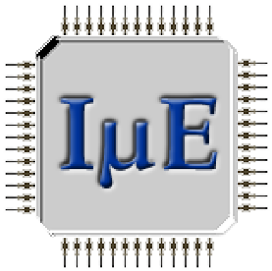
Karl Rupp

https://karlrupp.net/

*now:*
Freelance Scientist

*formerly:*
Institute for Microelectronics, TU Wien

in collaboration with colleagues at TU Wien:
J. Weinbub, F. Rudolf, A. Morhammer, T. Grasser, A. Jüngel
...and in discussion with colleagues all over the world

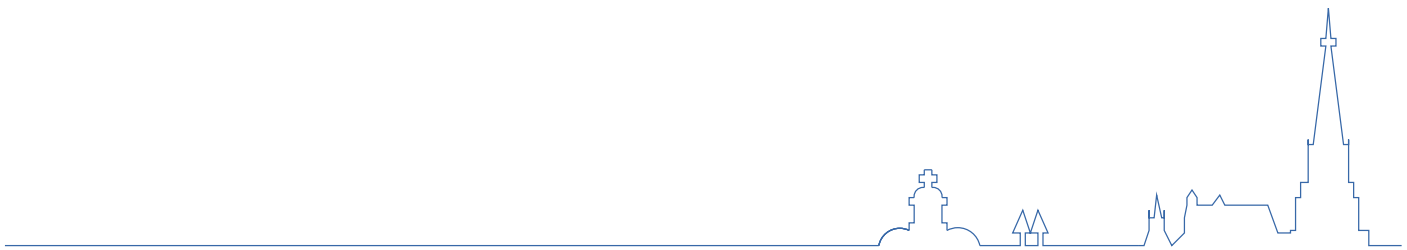



MS35 - To Thread or Not To Thread
SIAM PP, Paris
April 13, 2016

# Overview

Part 1: Why bother about MPI+X?

Part 2: What about X = threads?



## The Big Picture

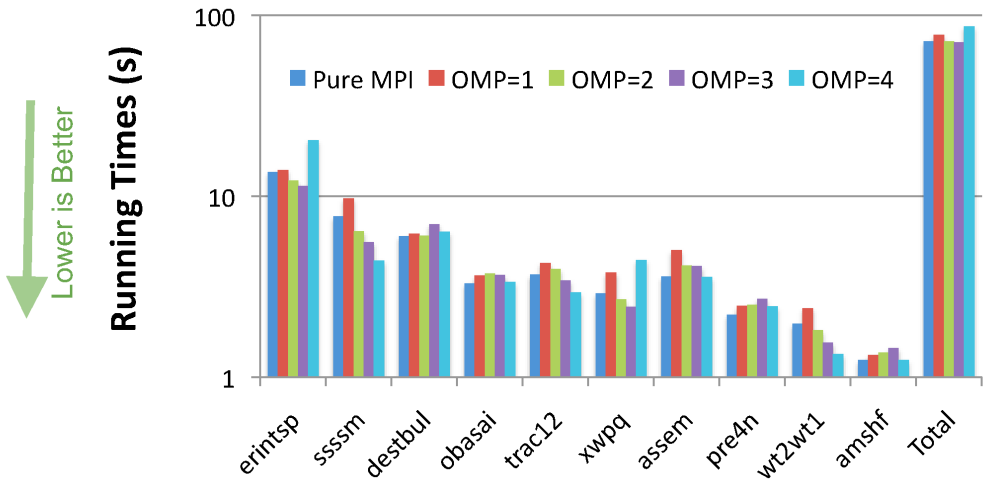
- **The next large NERSC production system "Cori" will be Intel Xeon Phi KNL (Knights Landing) architecture:**
  - >60 cores per node, 4 hardware threads per core
  - Total of >240 threads per node
- **Your application is very likely to run on KNL with simple port, but high performance is harder to achieve.**
- **Many applications will not fit into the memory of a KNL node using pure MPI across all HW cores and threads because of the memory overhead for each MPI task.**
- **Hybrid MPI/OpenMP is the recommended programming model, to achieve scaling capability and code portability.**
- **Current NERSC systems (Babbage, Edison, and Hopper) can help prepare your codes.**



"OpenMP Basics and MPI/OpenMP Scaling", Yun He, NERSC, 2015

# The Big Picture

## NWChem FMC, Add OpenMP to HotSpots (OpenMP #1)




- Total number of MPI ranks=60; OMP=N means N threads per MPI rank.
- Original code uses a shared global task counter to deal with dynamic load balancing with MPI ranks
- Loop parallelize top 10 routines in TEXAS package (75% of total CPU time) with OpenMP. Has load-imbalance.
- OMP=1 has overhead over pure MPI.
- OMP=2 has overall best performance in many routines.





"OpenMP Basics and MPI/OpenMP Scaling", Yun He, NERSC, 2015

## Summary (1)

- OpenMP is a fun and powerful language for shared memory programming.
- Hybrid MPI/OpenMP is recommended for many next generation architectures (Intel Xeon Phi for example), including NERSC-8 system, Cori.
- You should explore to add OpenMP now if your application is flat MPI only.



"OpenMP Basics and MPI/OpenMP Scaling", Yun He, NERSC, 2015

# Future Systems

## ASCR Computing Upgrades at a Glance

| System attributes | NERSC Now | OLCF Now | ALCF Now | NERSC Upgrade | OLCF Upgrade | ALCF Upgrade |
|---|---|---|---|---|---|---|
| Name/Planned Installation | **Edison** | **TITAN** | **MIRA** | **Cori 2016** | **Summit 2017-2018** | **Aurora 2018-2019** |
| System peak (PF) | 2.4 | 27 | 10 | >30 | 150 | >150 |
| Peak Power (MW) | 3 | 8.2 | 4.8 | <3.7 | 10 | ~13 |
| System memory per node | 64 GB | 38 GB | 16 GB | 64-128 GB DDR4 16 GB High Bandwidth | > 512 GB (High Bandwidth memory and DDR4) | TBA |
| Node performance (TF) | 0.460 | 1.452 | 0.204 | >3 | >40 | >15 times Mira |
| Node processors | Intel Ivy Bridge | AMD Opteron Nvidia Kepler | 64-bit PowerPC A2 | Intel Knights Landing many core CPUs Intel Haswell CPU in data partition | Multiple IBM Power9 CPUs & mulitple Nvidia Voltas GPUS | TBA |
| System size (nodes) | 5,200 nodes | 18,688 nodes | 49,152 | 9,300 nodes 1,900 nodes in data partition | ~3,500 nodes | ~50,000 nodes |
| System Interconnect | Aries | Gemini | 5D Torus | Aries | Dual Rail EDR-IB | TBA |
| File System | 17.6 PB, 168 GBs, Lustre® | 32 PB, 1 TB/s, Lustre® | GPFS™ | 28 PB, 744 GB/sec , Lustre® | 120 PB, 1 TB/s, GPFS™ | TBA |

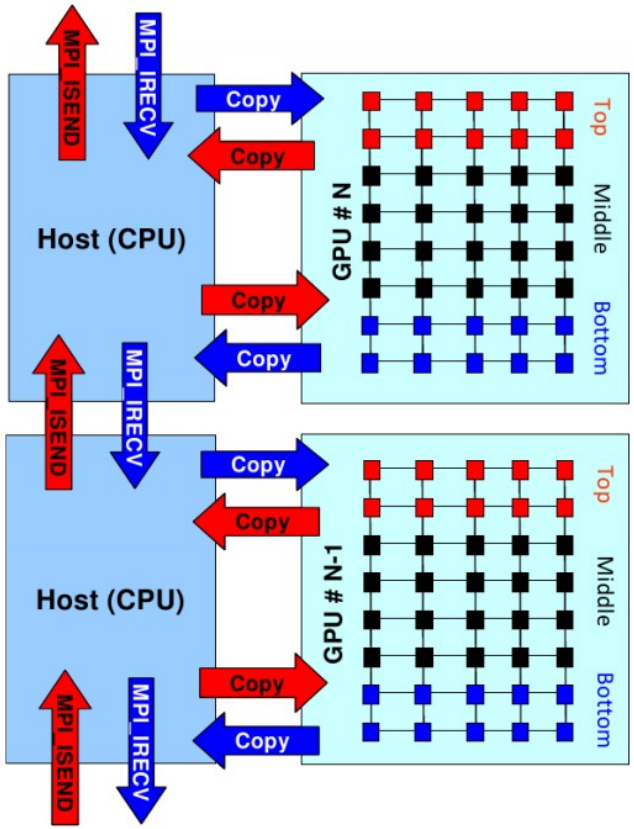
# Future Systems

## Some Systems are Hybrid

- Multi-core CPUs + many-core GPUs
- Multi-core CPUs + FPGAs?
- Multi-core CPUs + custom accelerators?

## Why MPI+X for Hybrid Systems?

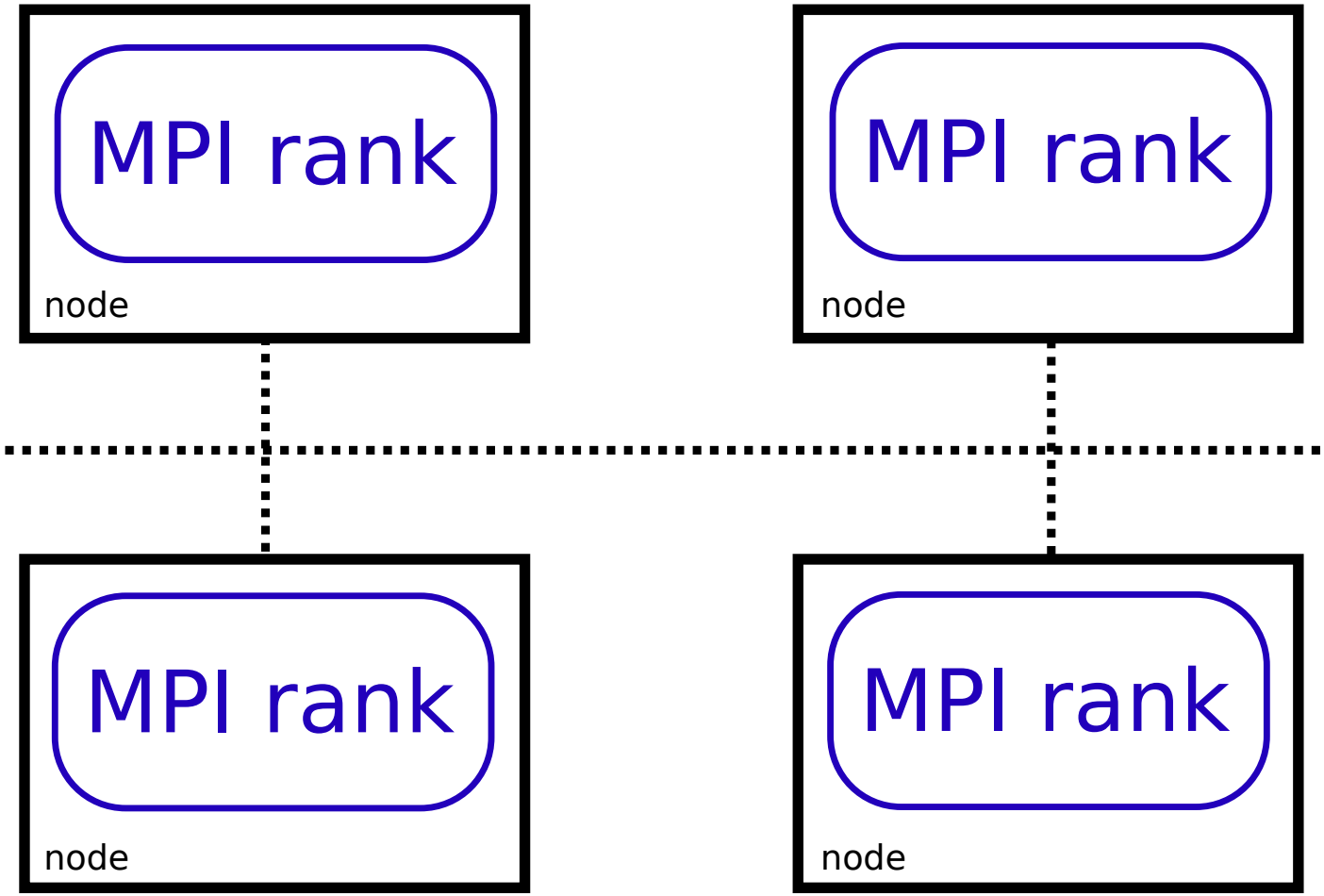
- MPI not available on GPUs/FPGAs/accelerators




http://geco.mines.edu/tesla/cuda_tutorial_mio/pic/mpi_cuda.jpg

NVIDIA CUDA OpenMP OpenCL

# Simple MPI Model




## Hybridization Easy?

```
#pragma omp parallel for
```

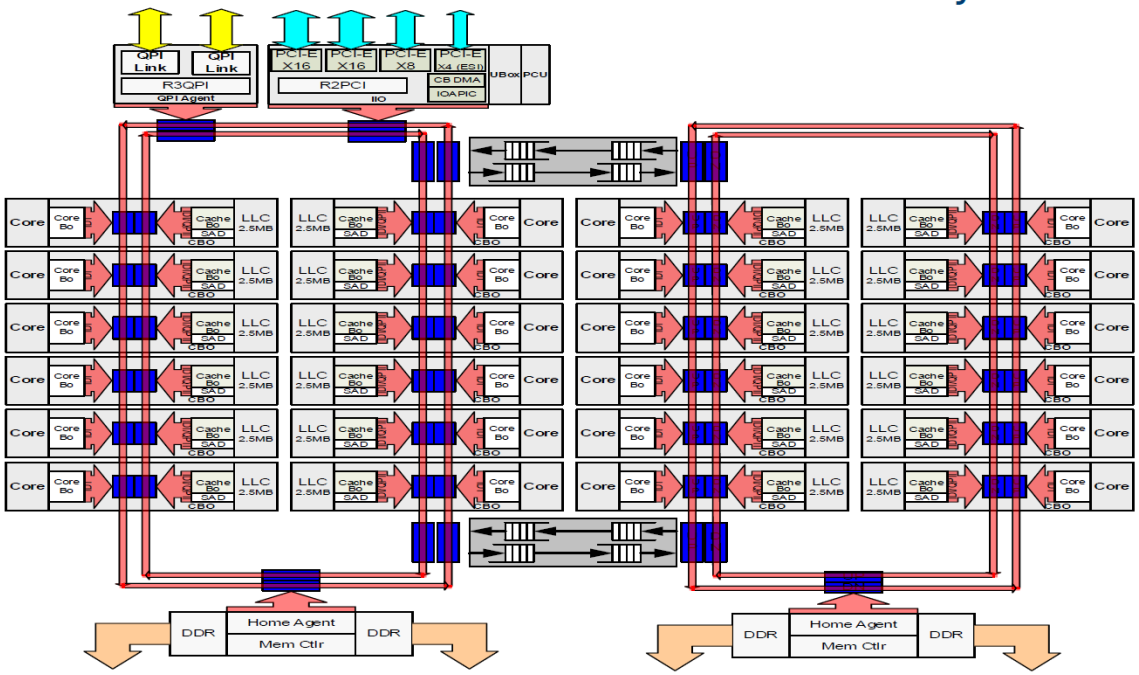
# Simple MPI Model

## Mind the Details

NUMA domains: Data locality matters

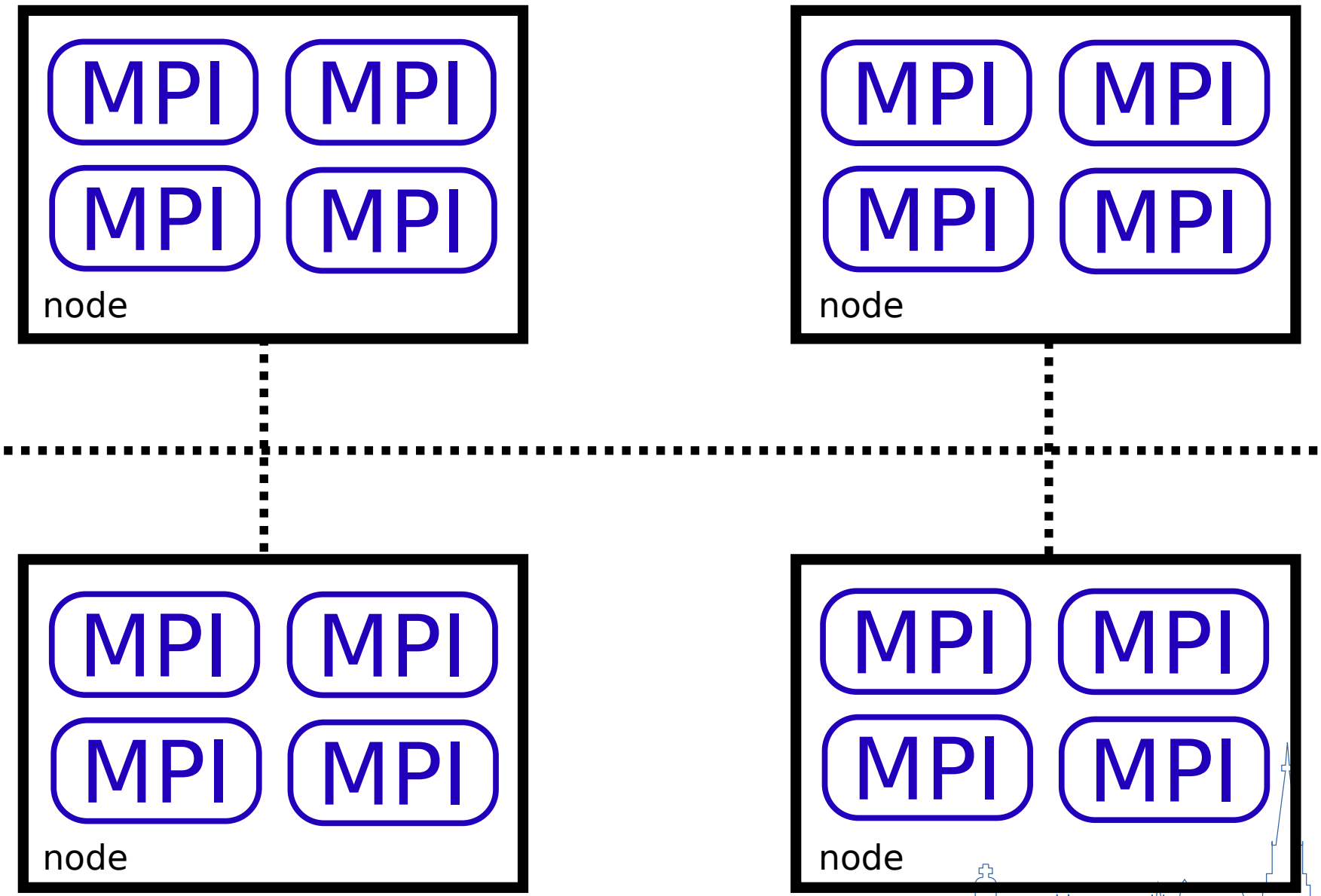
CPU internals: Ring buses

Intel® Xeon® Processor E5 v4 Product Family HCC




http://images.anandtech.com/doci/10158/v4_24coresHCC.png

# NUMA-aware MPI Model



# Hybrid MPI Model

## Hybridization

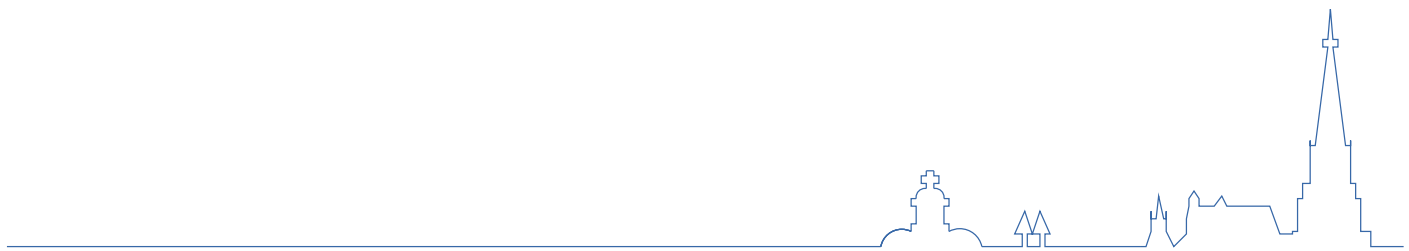
One or multiple multi-core CPUs

One or multiple GPUs/FPGAs/accelerators

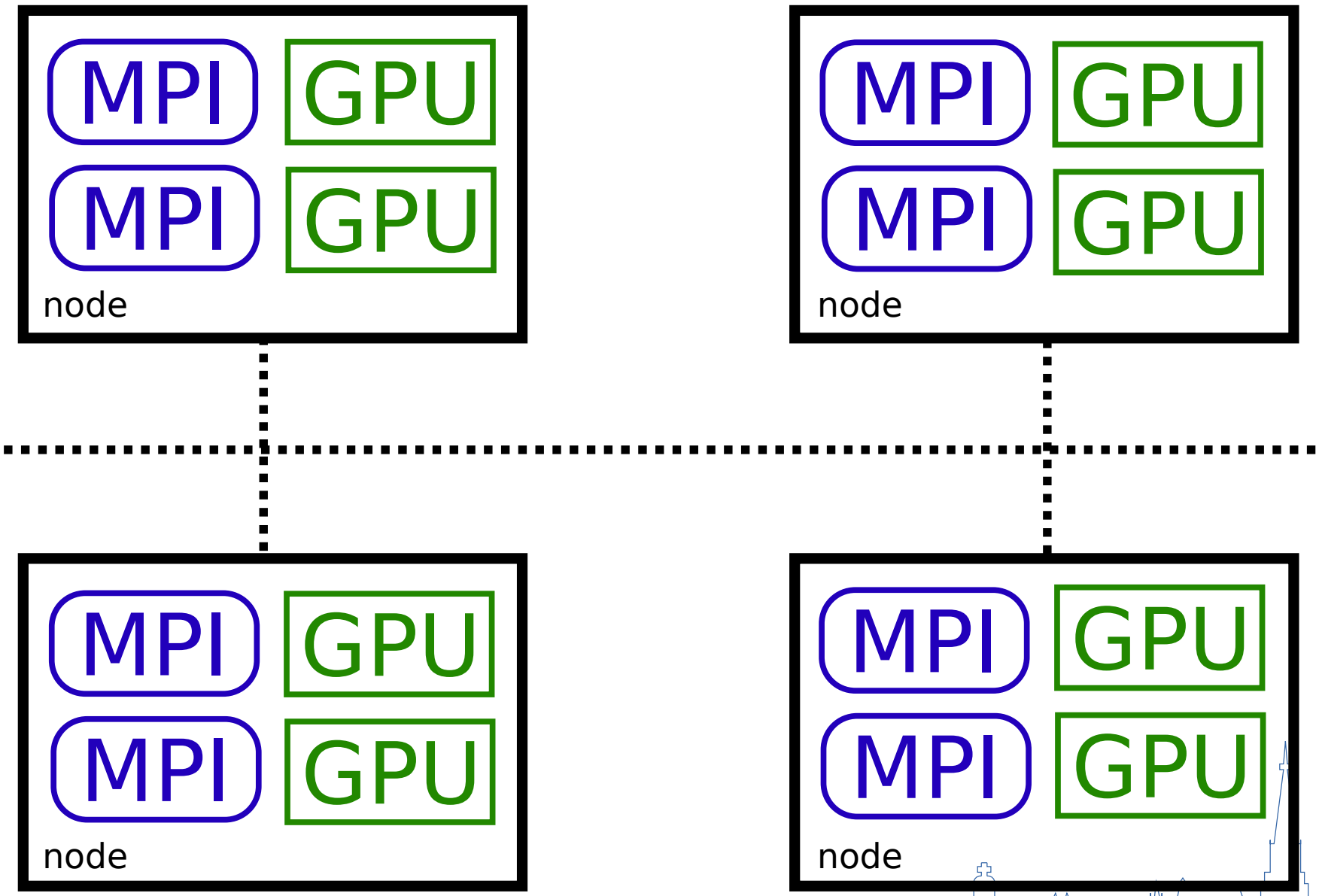
## CPU Starvation

Use one MPI rank per GPU (shepard process)

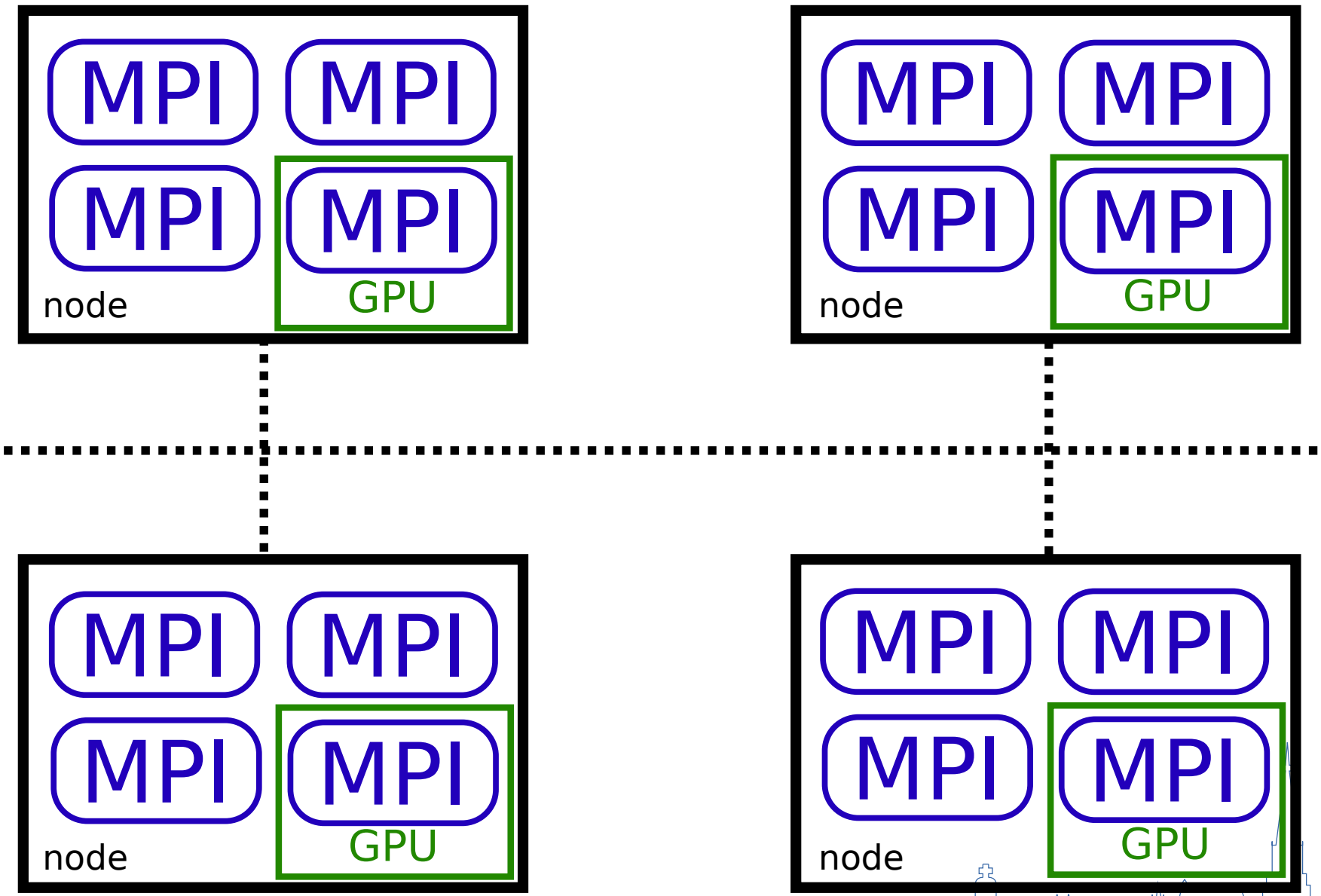
Problem: Waste of CPU resources



# GPUs in the MPI Model



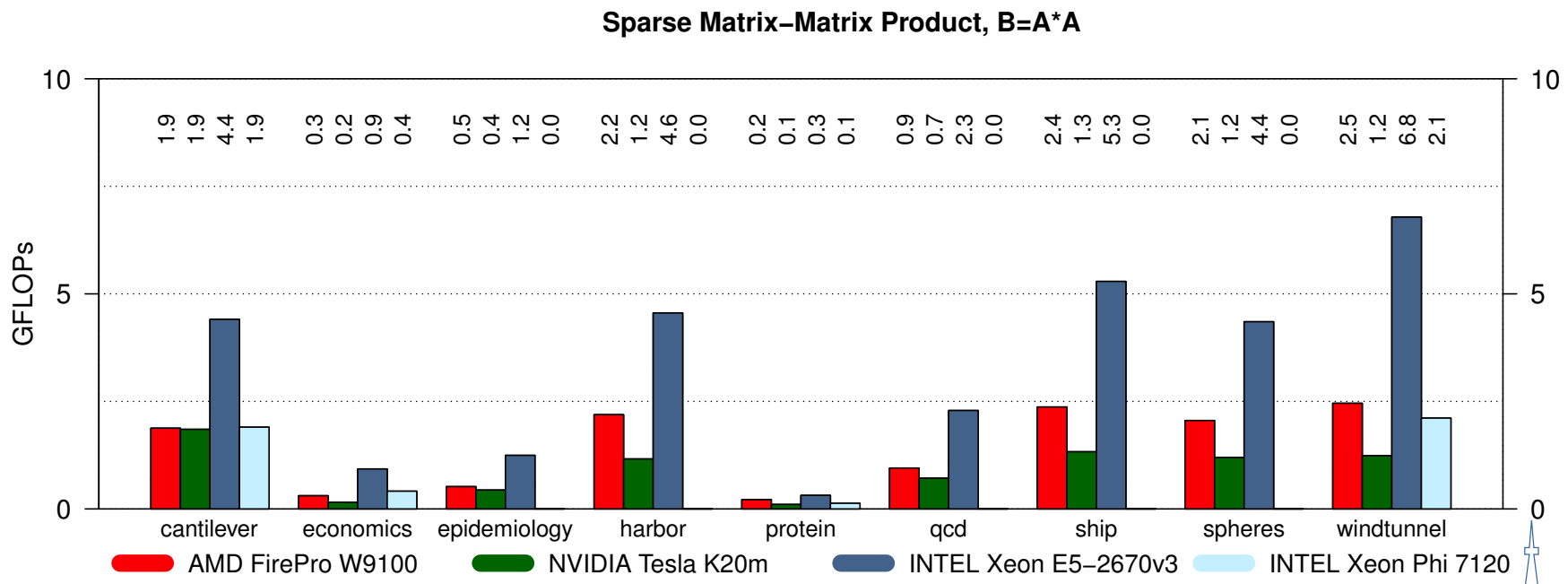
# GPUs in the MPI Model



## MPI+X for Hybrid Systems

Important question: What to compute where?

Unimportant question: What is X?



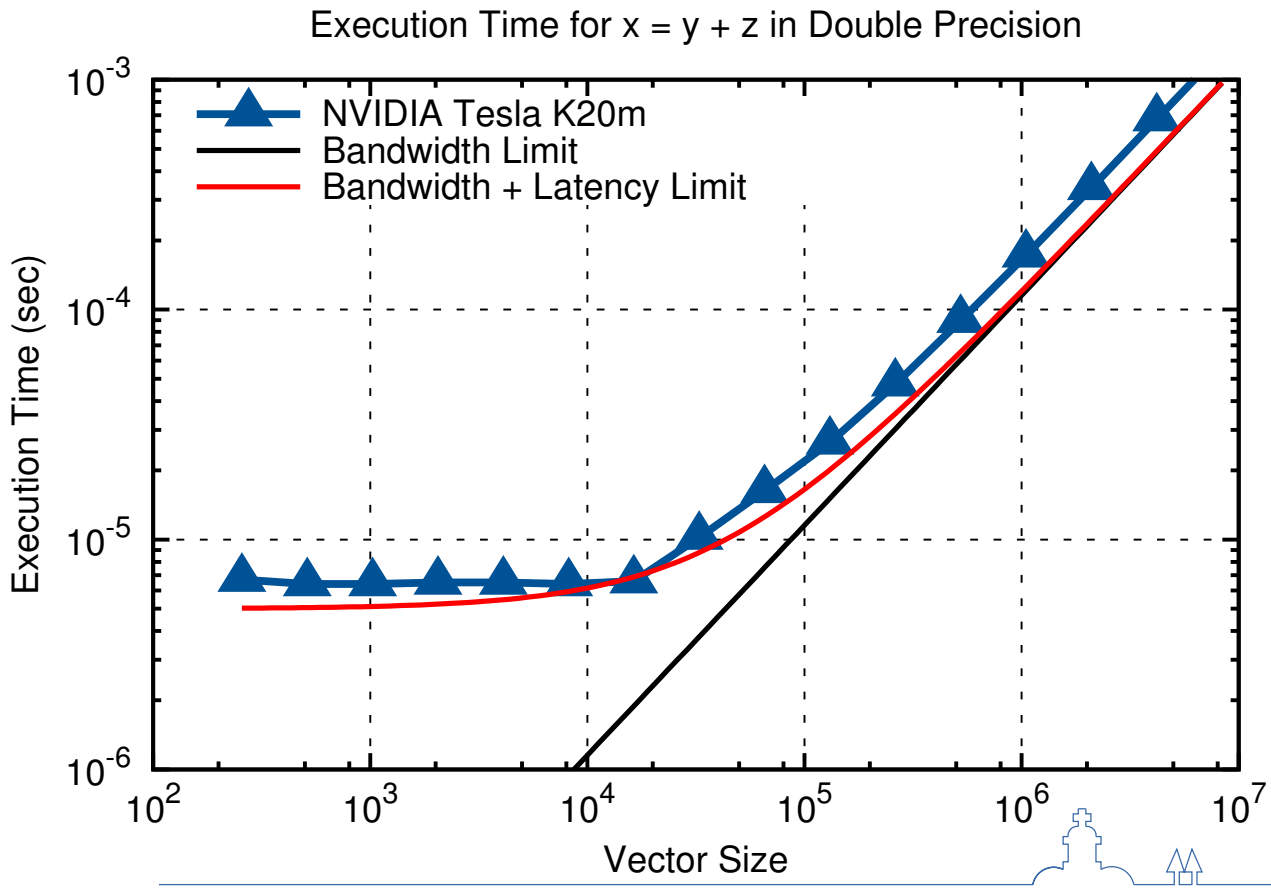
# PCI-Express Bottleneck

## Vector Addition

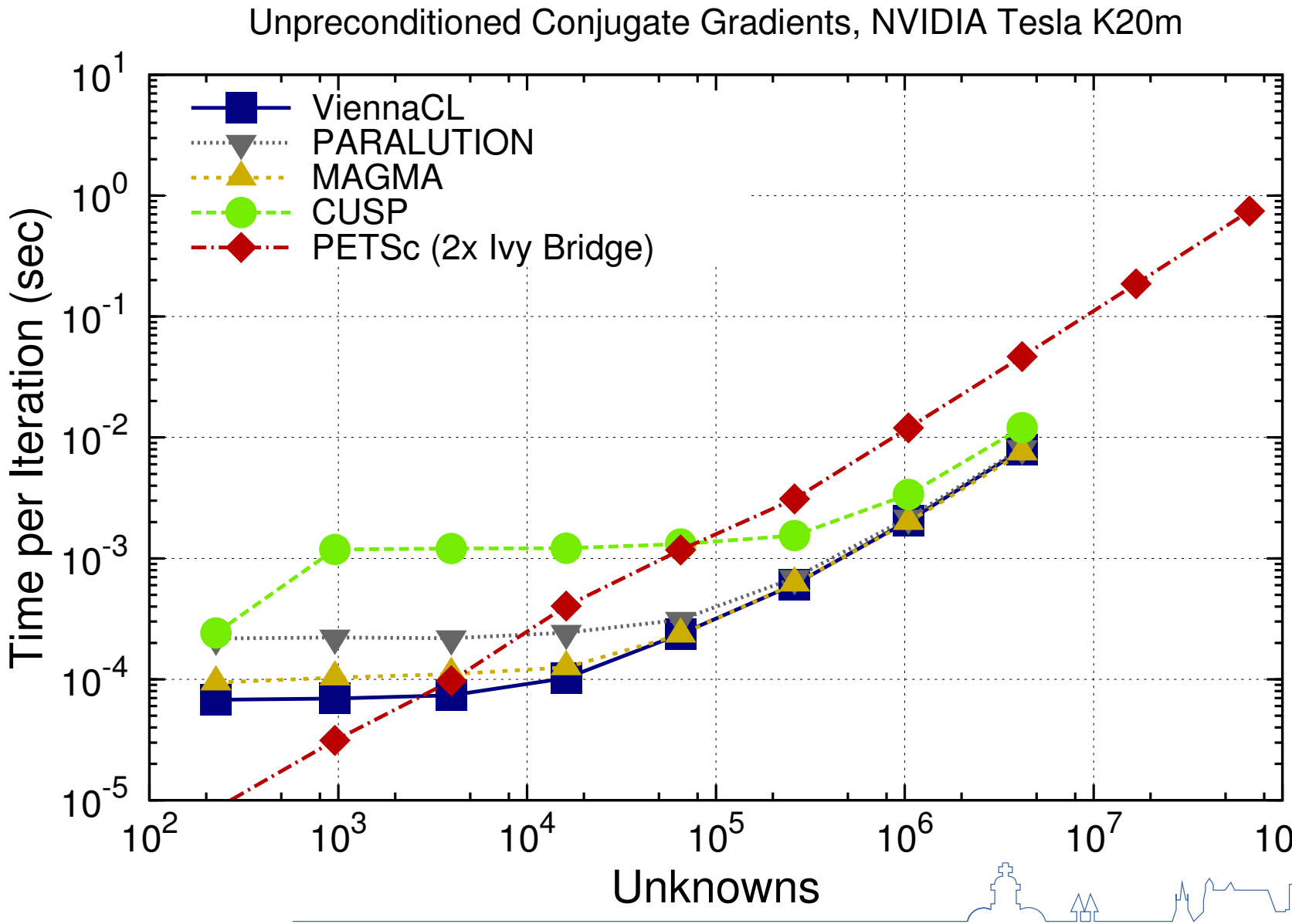
$x = y + z$ with $N$ elements each

1 FLOP per 24 byte in double precision

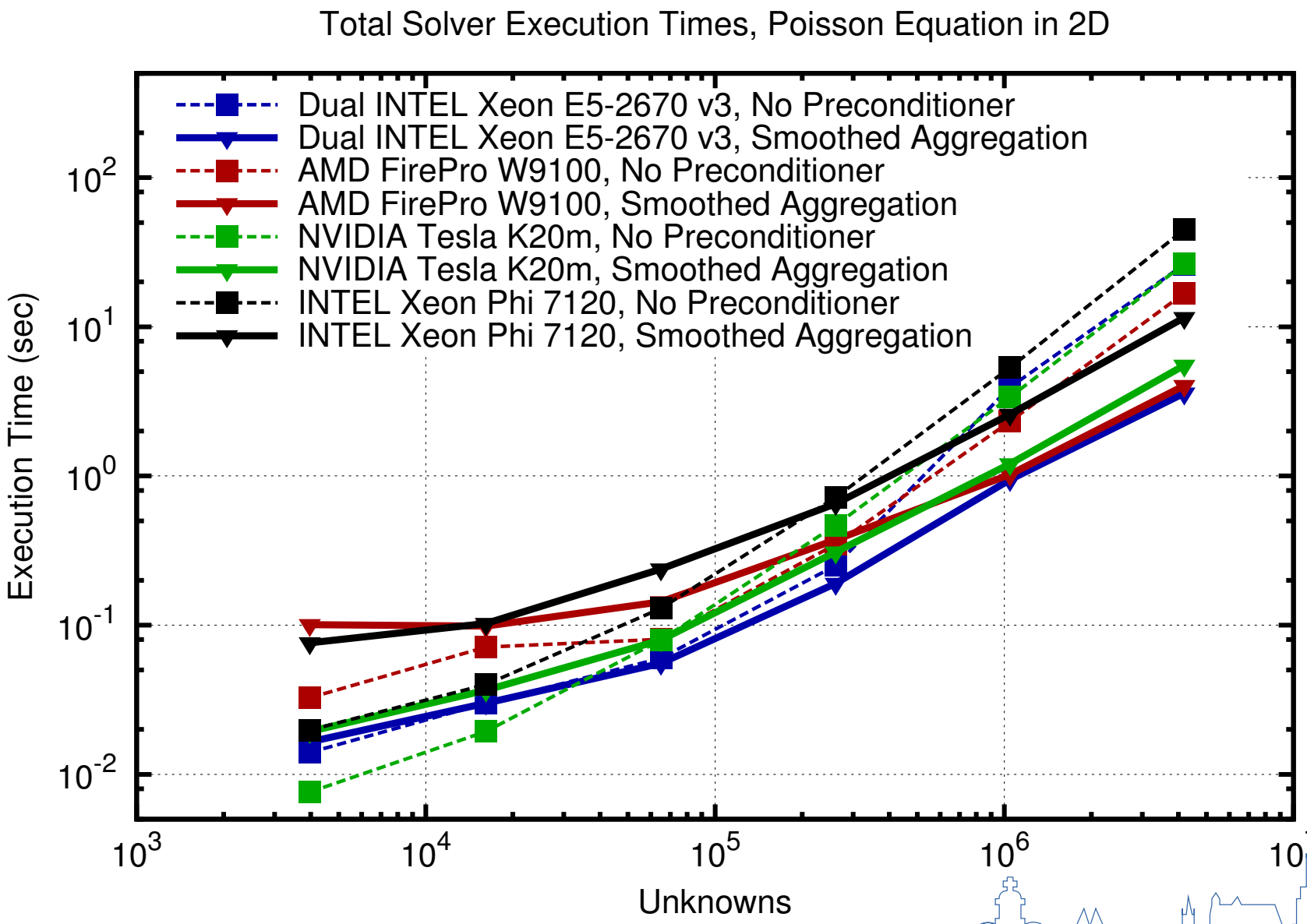
Limited by memory bandwidth $\Rightarrow T_2(N) \overset{?}{\approx} 3 \times 8 \times N/\text{Bandwidth} + \text{Latency}$



# PCI-Express Bottleneck



# PCI-Express Bottleneck



# PCI-Express Bottleneck

## Example: Algebraic Multigrid

SpGEMM on CPU (AMG setup)

SpMV & friends on GPUs (AMG solve)

## Model 1: Side-by-Side

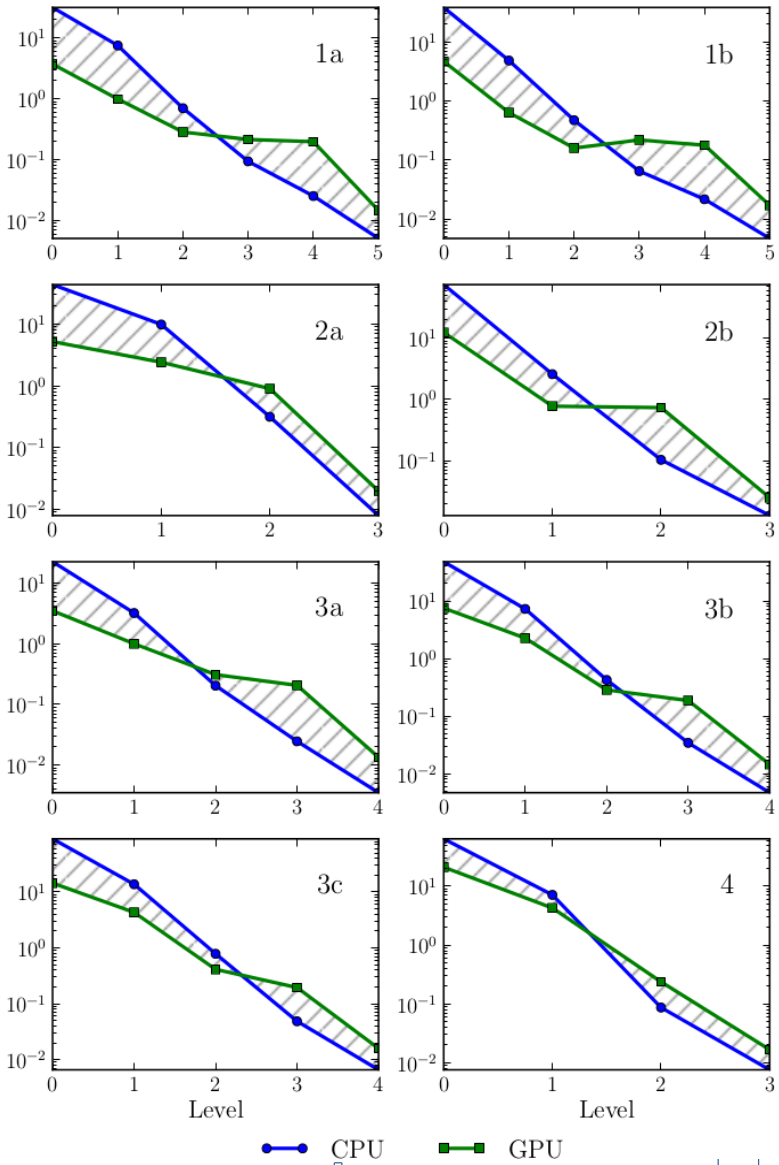
Use MPI ranks for CPU

How to decompose problem?

## Model 2: GPU shepards only

Overlapping of GPU and CPU work within rank

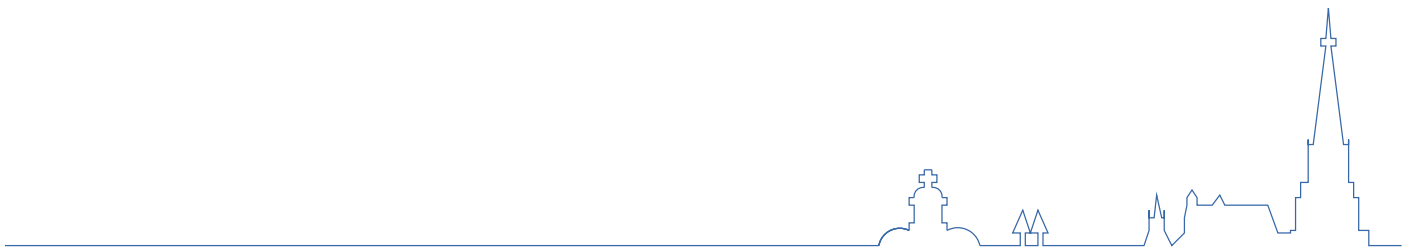
Reimplement message passing on MPI rank level?



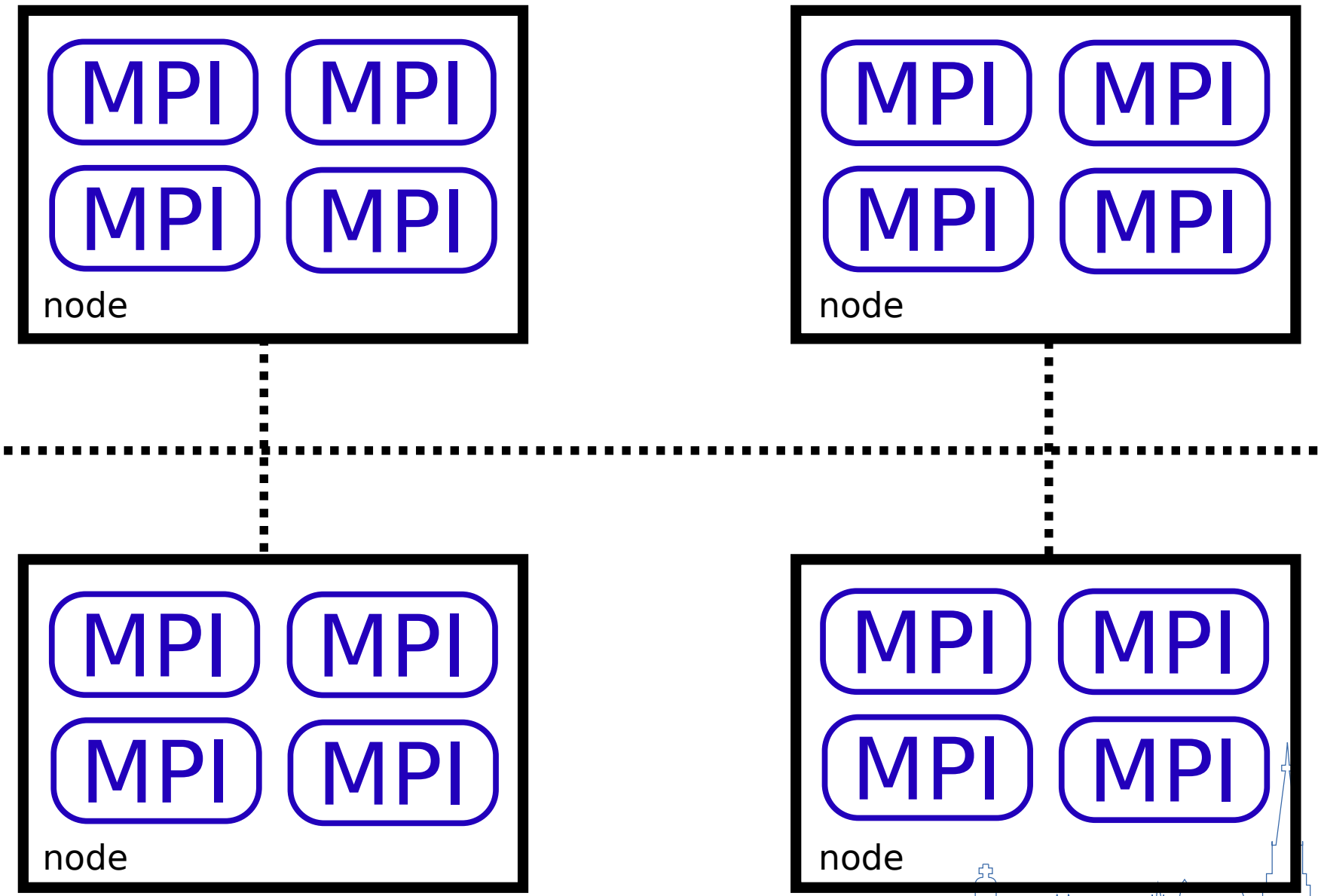

N. Bell *et al.*, SISC 34(4), 2012

# Overview

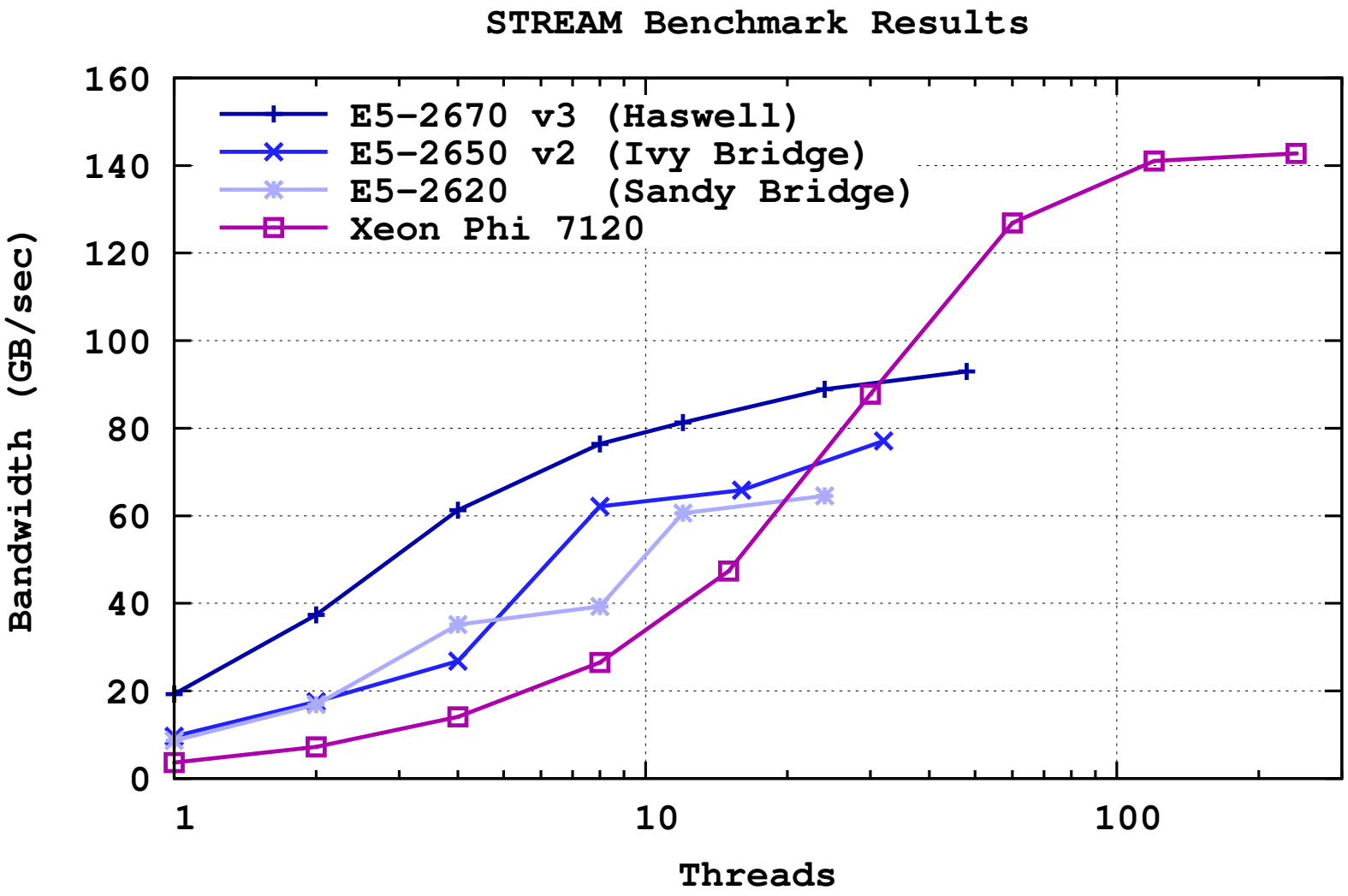
Part 2: What about X = threads?



# NUMA-aware MPI Model



# Memory Bandwidth vs. Parallelism




https://www.karlrupp.net/2015/02/stream-benchmark-results-on-intel-xeon-and-xeon-phi/

# Threads and Library Interfaces

## Attempt 1

Library spawns threads

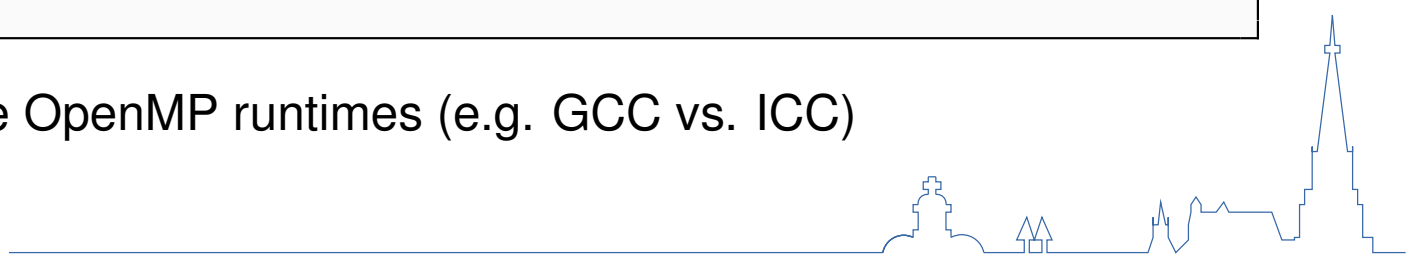
```
void library_func(double *x, int N) {
  #pragma omp parallel for
  for (int i=0; i<N; ++i) x[i] = something_complicated();
}
```

## Problems

Call from multi-threaded environment?

```
void user_func(double **y, int N) {
  #pragma omp parallel for
  for (int j=0; j<M; ++j) library_func(y[j], N);
}
```

Incompatible OpenMP runtimes (e.g. GCC vs. ICC)



# Threads and Library Interfaces

## Attempt 2

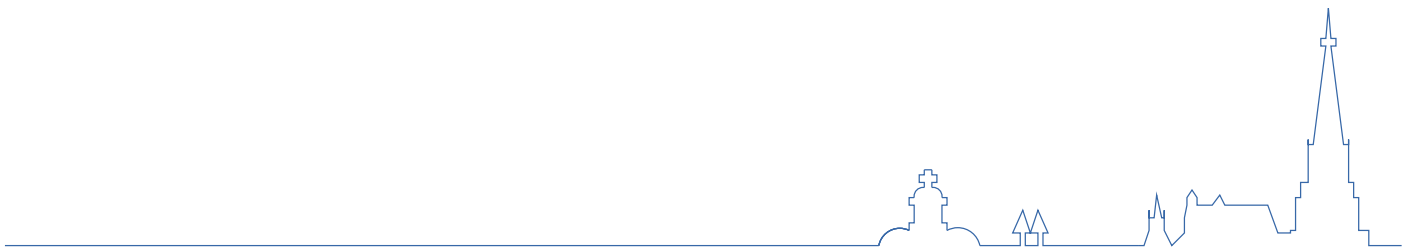
Use pthreads/TBB/etc. instead of OpenMP to spawn threads

Fixes incompatible OpenMP implementations (probably)

## Problems

Still a problem with multi-threaded user environments

```
void user_func(double **y, int N) {
  #pragma omp parallel for
  for (int j=0; j<M; ++j) library_func(y[j], N);
}
```



## Attempt 3

Hand back thread management to user

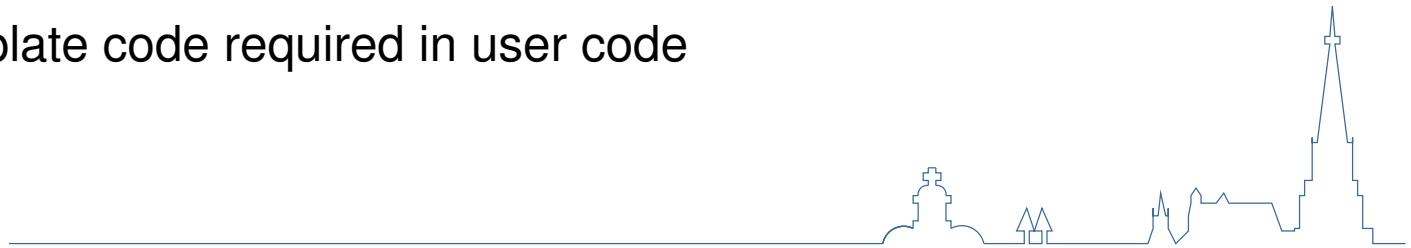
```
void library_func(ThreadInfo ti, double *x, int N) {
  int start = compute_start_index(ti, N);
  int stop  = compute_stop_index(ti, N);
  for (int i=start; i<stop; ++i)
    x[i] = something_complicated();
}
```

## Implications

Users can use their favorite threading model

API requires one extra parameter

Extra boilerplate code required in user code



# Threads and Library Interfaces

## Reflection

Extra thread communication parameter

```
void library_func(ThreadInfo ti, double *x, int N) {...}
```

Rename thread management parameter

```
void library_func(Thread_Comm c, double *x, int N) {...}
```

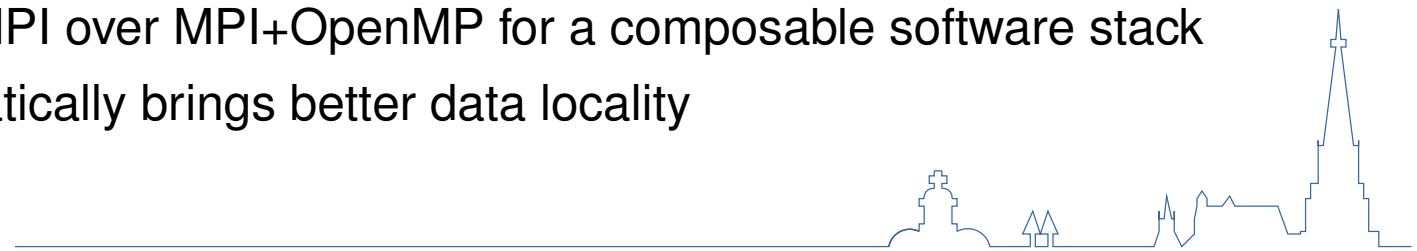
Compare:

```
void library_func(MPI_Comm comm, double *x, int N) {...}
```

## Conclusion

Prefer flat MPI over MPI+OpenMP for a composable software stack

MPI automatically brings better data locality



# Summary

## MPI+X Opportunities

- CPUs for sequential potions
- Non-CPUs for fine-grained parallel potions
- Get funding for reimplementing existing things

## MPI+X Challenges

- Use all components of a hybrid system
- Productivity?
- Scientific progress?