OP2 FOR MANY-CORE ARCHITECTURES

G.R. Mudalige, M.B. Giles,

Oxford e-Research Centre, University of Oxford

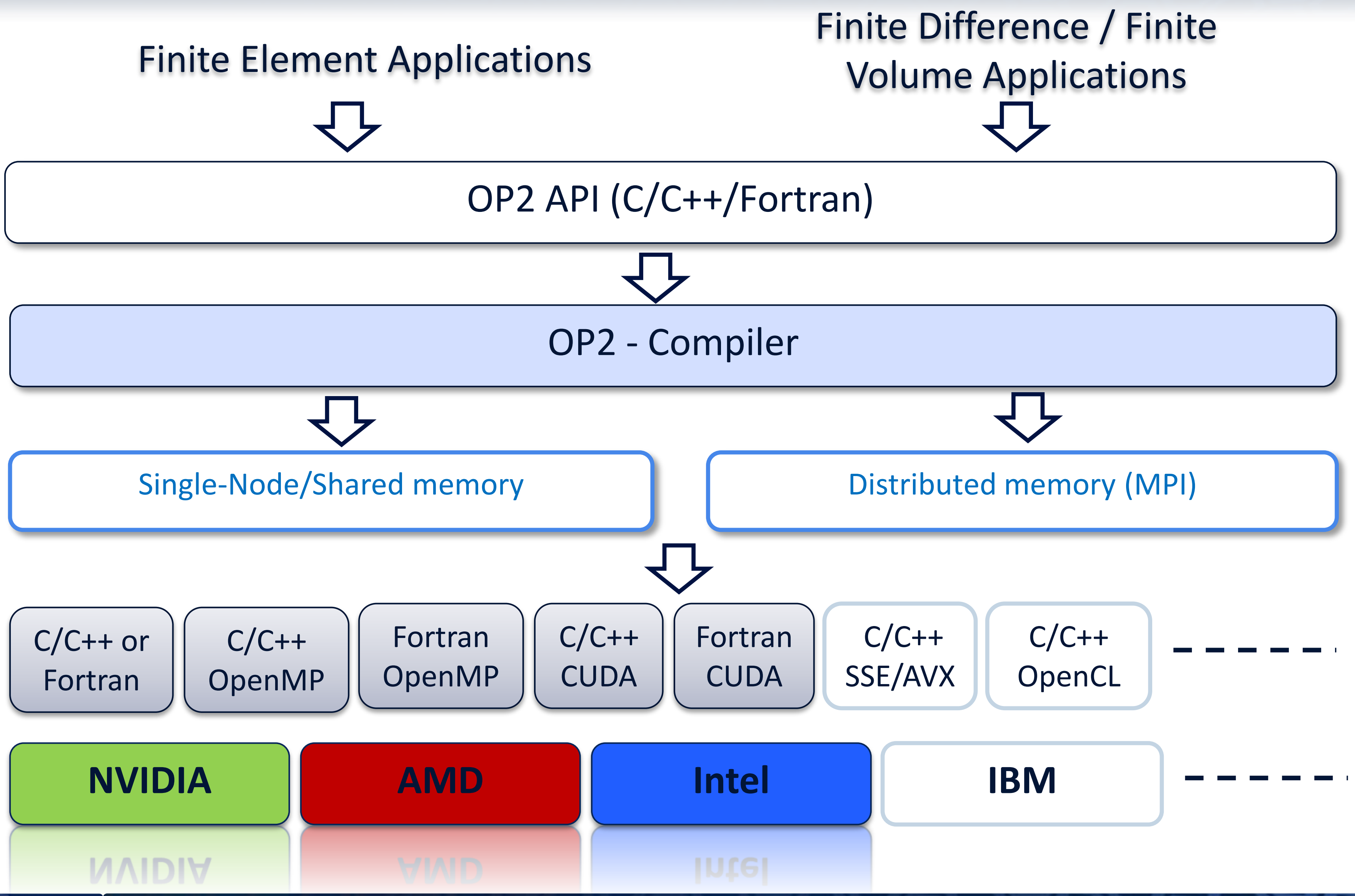
gihan.mudalige@oerc.ox.ac.uk

27th Jan 2012

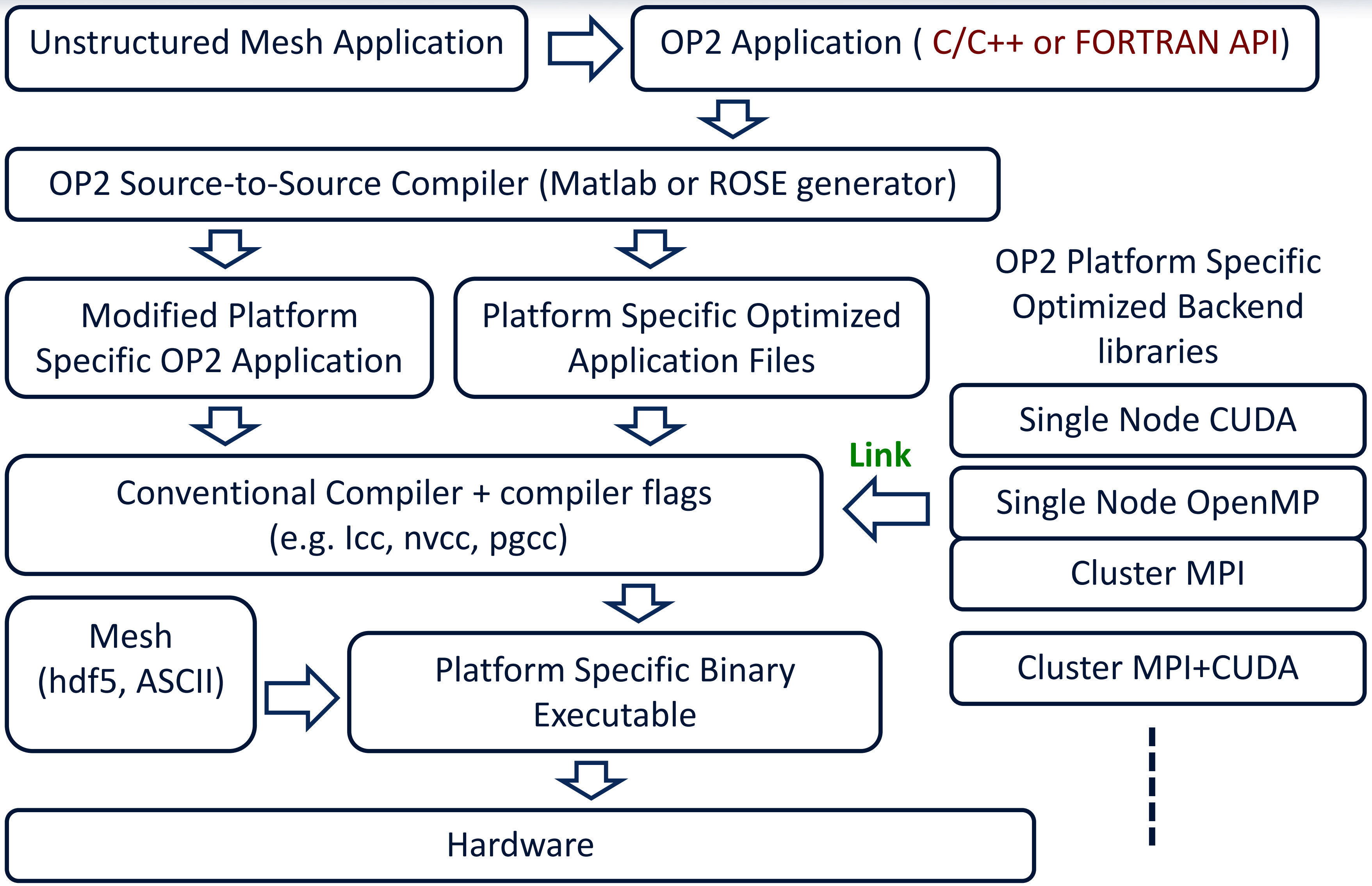
AGENDA

- ❑ OP2 Current Progress
- ❑ Future work for OP2
- ❑ EPSRC proposal for extending OP2 for structured grids
- ❑ OP2 for AWE applications
- ❑ Funding or support opportunities to be explored

The OP2 Framework



OP2 – Generating Platform-Specific Executables


Unstructured Mesh Application
OP2 Application (C/C++ or FORTRAN API)
OP2 Source-to-Source Compiler (Matlab or ROSE generator)
Modified Platform Specific OP2 Application
Platform Specific Optimized Application Files
OP2 Platform Specific Optimized Backend libraries
Single Node CUDA
Single Node OpenMP
Cluster MPI
Cluster MPI+CUDA
Link
Conventional Compiler + compiler flags (e.g. Icc, nvcc, pgcc)
Mesh (hdf5, ASCII)
Platform Specific Binary Executable
Hardware

OP2 – Features

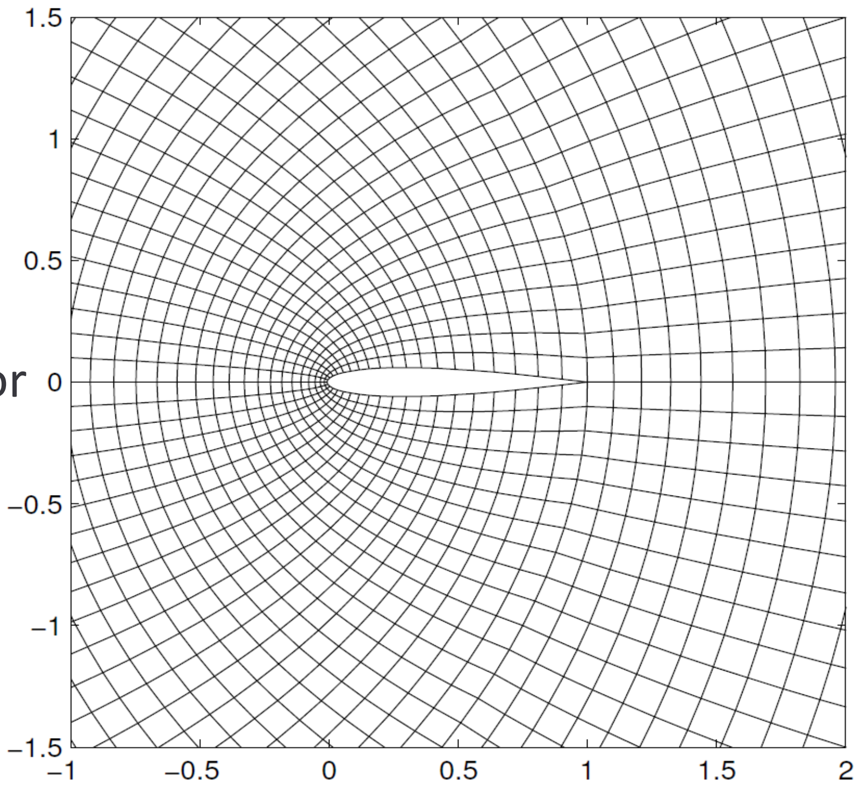
- Parallel file I/O using HDF5
- Partitioning routines from ParMetis and PT-Scotch
  - geometric partitioning
  - k-way partitioning
- Implicit diagnostics and performance monitoring
  - Parallel loop runtime and bandwidth utilization
  - partition and halo sizes per process
  - message sizes, communication frequency and number of neighbours communicated per process
- Automatic Check pointing – *to be implemented*

OP2 –Progress to Date

- Currently Supports five back-ends:
  - Single-threaded on a CPU
  - Multi-threaded on a CPU using OpenMP
  - Single GPU node using CUDA
  - Distributed memory CPU cluster using MPI
  - Distributed memory GPU cluster using MPI+CUDA
- Experimental (under testing):
  - Cluster of multi-threaded CPUs using MPI and OpenMP
  - Single GPU node using OpenCL
  - Single multi-core CPU with AVX

Example Application - Airfoil

- A non-linear 2D inviscid airfoil code
  - 2D Euler equations
  - cell centred finite volume method with scalar dissipation
- Representative of the 3D viscous flow calculations we eventually want to do for Rolls Royce's Hydra CFD application
- We investigate two mesh sizes
  - 720K nodes, 720K cells and 1.5 M edges.
  - 13 M nodes, 13 M cells and 26 M edges
- Consists of five parallel loops
  - save_soln and update – direct loops
  - adt_calc, res_calc, bres_calc – indirect loops



- The most compute intensive loop (`res_calc`) is called 2000 times, in each loop iteration an edge performs 100 floating-point operations

OP2 – Single Node Performance (1.5 M edges)

Node System	Cores /node (Clk/core)	Mem. /node	Compiler + Compiler Flags	Run-time (secs)
2 × Intel Xeon X5650 (Westmere)	12 (2.67GHz)	24 GB	Intel C++ (11.1) -O3 -xSSE4.2	42.53 (24 OMP) 31.79 (22 MPI) 31.50 (11 MPI x 2 OMP)
2 × AMD Opteron 6276 (Interlagos)	32 (2.3GHz)	32 GB	-O3 -fastsse -Mipa=fast -Minline=levels:10	68.79(32 OMP) 43.83 (32 MPI) 23.63 (4 MPI x 8 OMP)
GeForce GTX560Ti	384 (1.6GHz)	1 GB ecc-off	nvcc (4.0) -O3 -arch=sm_20 -Dlcm=ca -use_fast_math	19.63 (CUDA)
Tesla C2050	448 (1.15 GHz)	3 GB ecc-on		19.40 (CUDA)
Tesla C2070	448 (1.15 GHz)	6 GB ecc-off		15.93 (CUDA)

OP2 – Single Node (OpenMP) Performance (1.5 M edges)

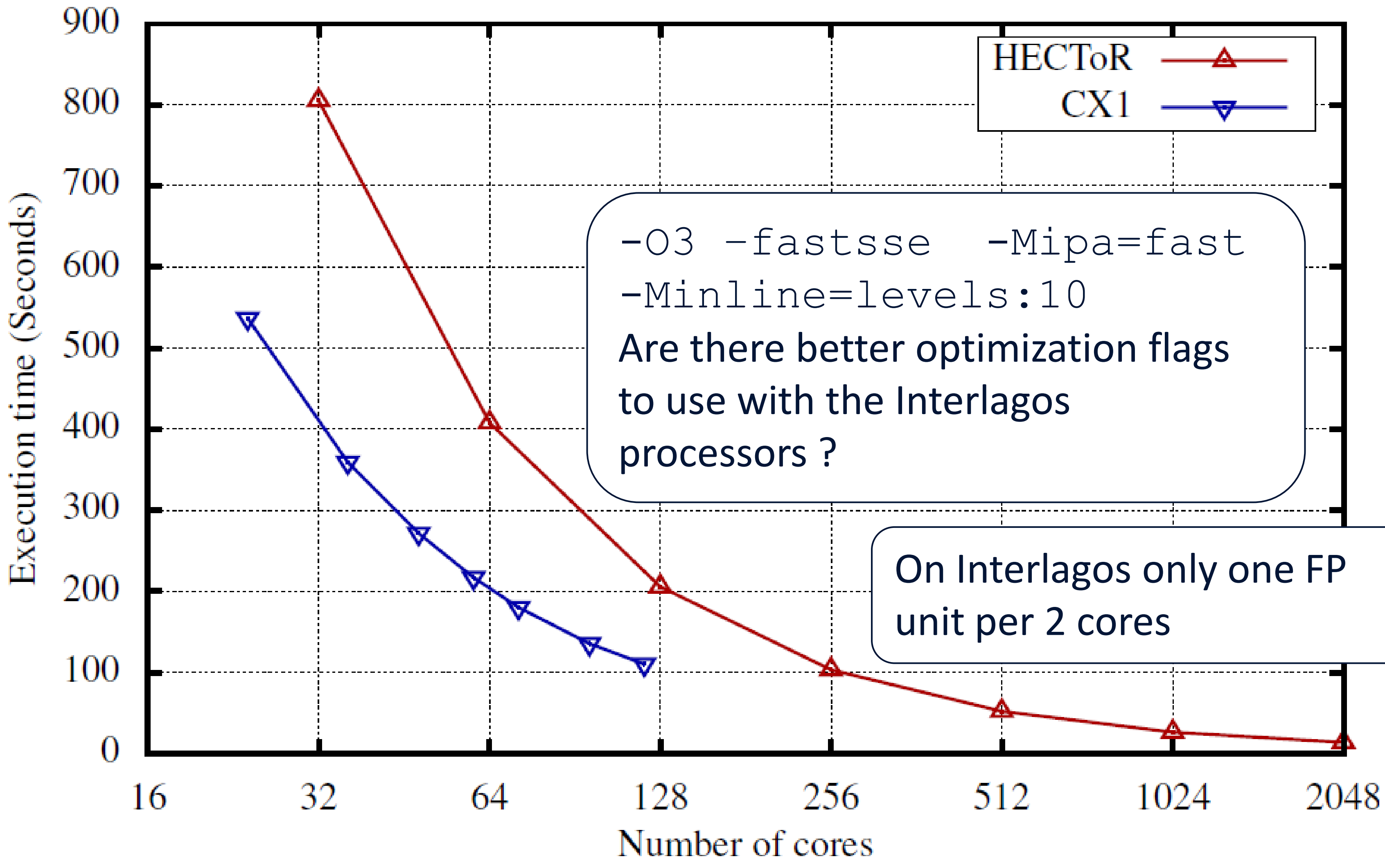
Achieved Floating-point and Bandwidth Performance for `res_calc`

Node System	`res_calc` time (sec)	**Achieved FP-rate** (GFlops /sec)	**Peak FP-rate** (GFlops /sec)	**Achieved Bandwidth** (GB/s)	**Peak Bandwidth** (GB/s)
2 × Intel Xeon X5650 (Westmere)	19.70	15	140	22.26	32
2 × AMD Opteron 6276 (Interlagos)	39.63	7.56	294	11.05	51
Tesla C2070	10.29	27.44	515	43.06	144

Cluster Specifications

System	HECToR (CrayXE6)	CX1 (Dell Cluster)	SkyNet (GPU Cluster)
Node Architecture	2x16-core AMD Opteron 2.3 GHz (Interlagos)	2x6-core Xeon X5650 2.67 GHz (Westmere)	2 x Tesla C2050 + 2 x Intel Xeon E5440 2.83 GHz
Cores/Node	32	12	2 GPUs + 8 CPU cores
Mem./Node	32GB	24GB	~6 GB (on GPUs) + 8 GB
Interconnect	Cray Gemini Interconnect	Dual QDR InfiniBand	DDR InfiniBand
O/S	CLE 4.0	RHEL 5.6	CentOS 5.6, Rocks 5.1
Compilers	PGI CC 11.9 Cray MPI	ICC 11.1 Intel MPI 3.1	ICC 12.0.0 OpenMPI 1.4.3
Compiler flags	-O3 -fastsse -Mipa=fast -Minline=levels:10	-O2 –xSSE4.2	-O2 -xSSE4.1 -arch=sm 20 -use fast math

Cluster Performance (MPI only) - 26M edges

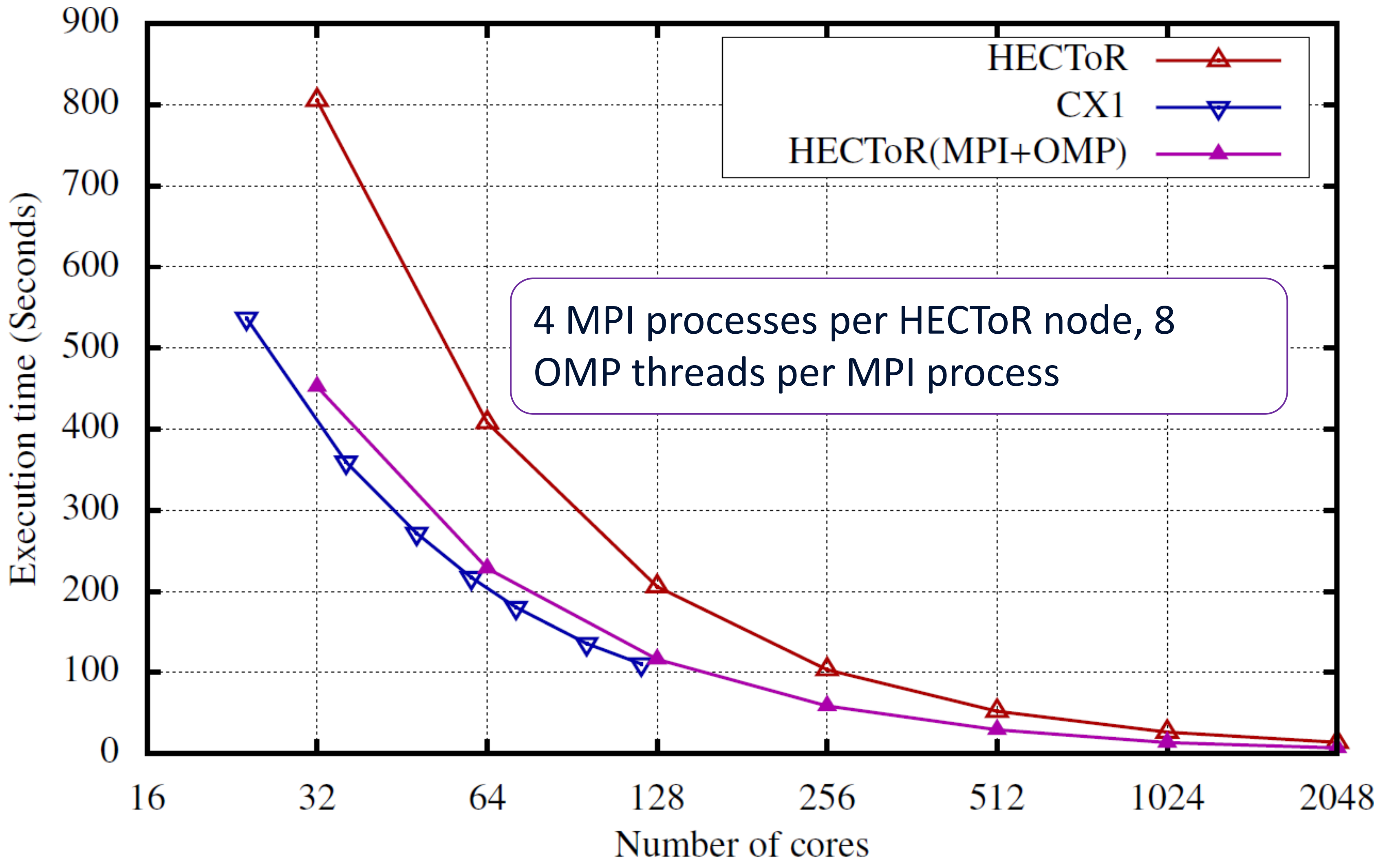


```
-O3 -fastsse  -Mipa=fast
-Minline=levels:10
```

Are there better optimization flags to use with the Interlagos processors ?

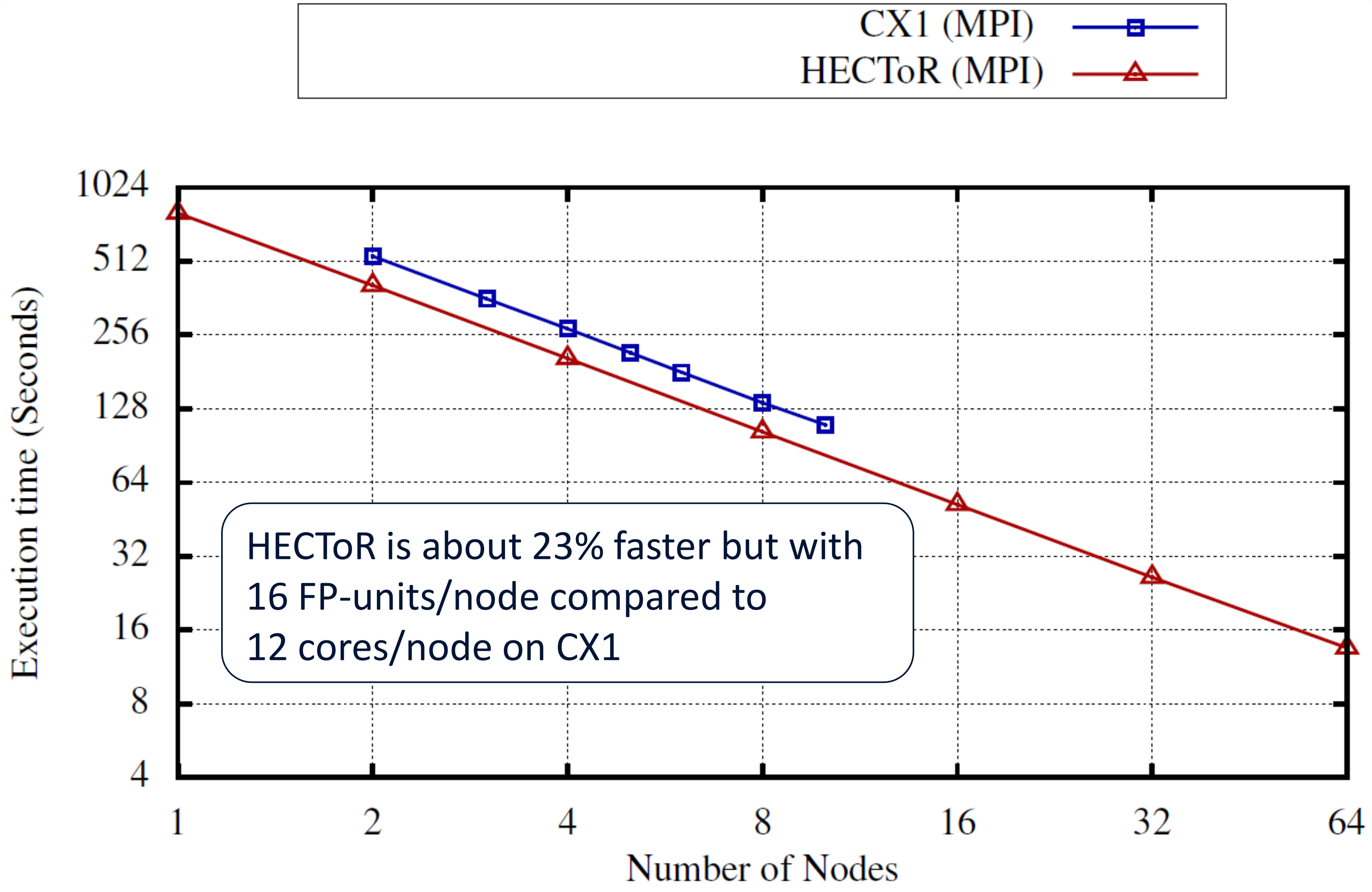
On Interlagos only one FP unit per 2 cores

Cluster Performance (MPI + OpenMP) - 26M edges



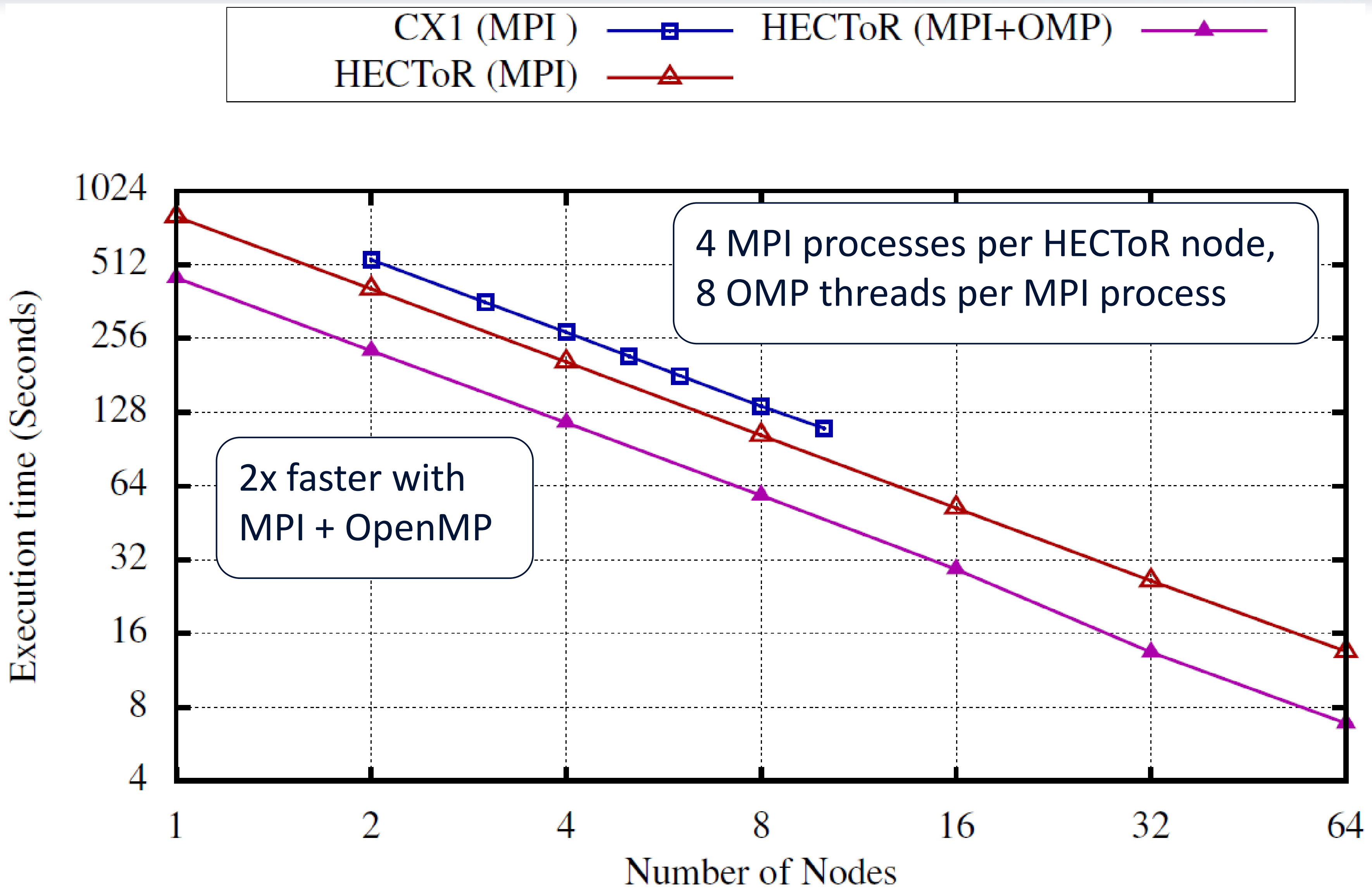
4 MPI processes per HECToR node, 8 OMP threads per MPI process

Cluster Performance (MPI only) - 26M edges

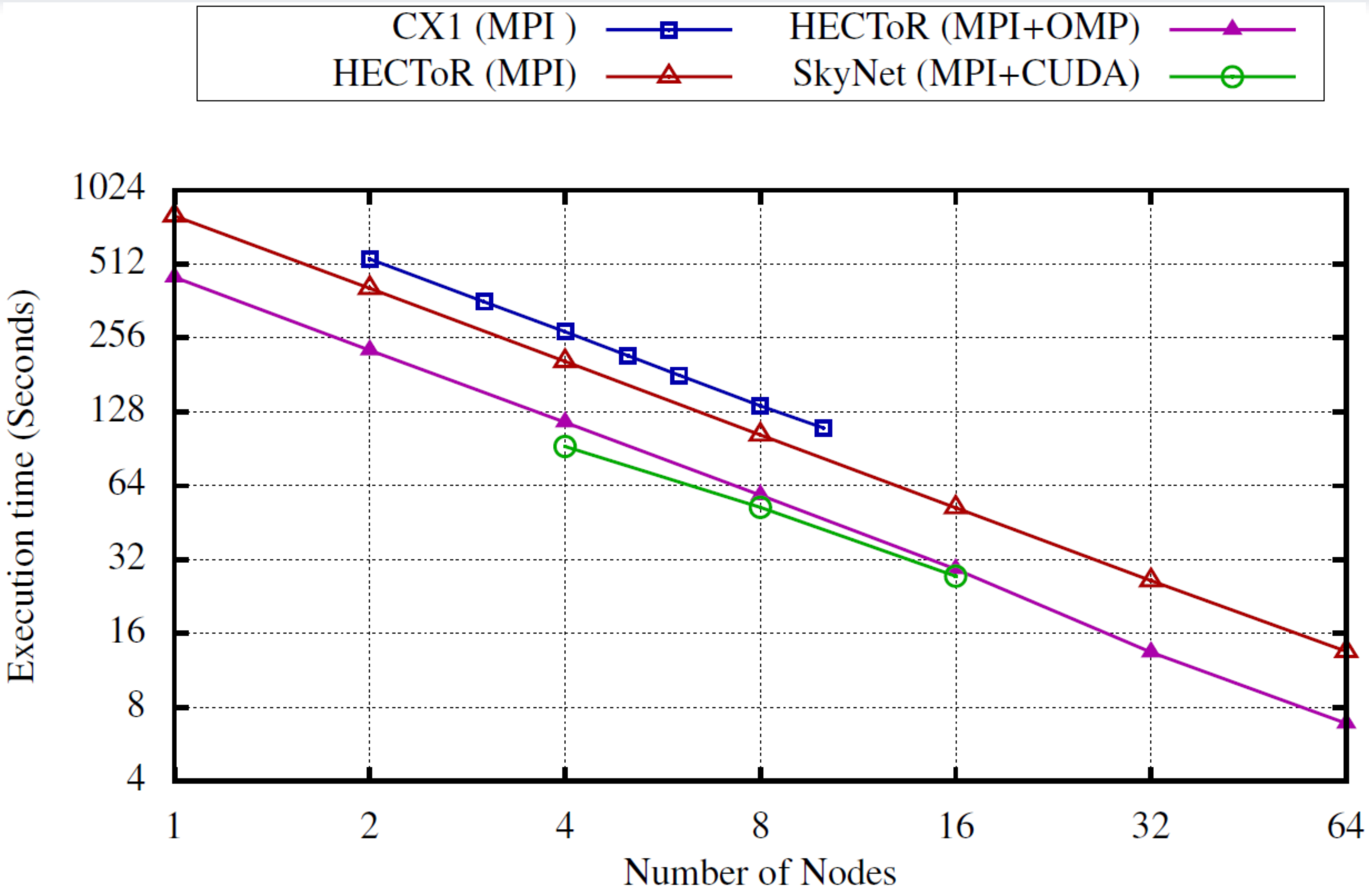


HECToR is about 23% faster but with 16 FP-units/node compared to 12 cores/node on CX1

Cluster Performance (MPI +OpenMP) - 26M edges



ManyCore Cluster Performance (26M edges)



SUMMARY

- Single node "pure" OpenMP performance getting worse as the number of cores per node (i.e. different NUMa regions) increase –
  - The interconnect between processor sockets is becoming a bottleneck ?

- Better performance to when OpenMP is combined with MPI
  - Must be careful of NUMa regions, otherwise performance could be worse than running pure MPI.

- NVIDIA GPU cluster performance (using MPI+CUDA) is almost equivalent to Cray XE6 performance (using MPI + OpenMP)

Download

- ❑Current OP2 source and Airfoil application + mesh available for download

  http://www.oerc.ox.ac.uk/research/op2

- ❑ OP2 development repository hosted at GIT-HUB

  https://github.com/carlobertolli/OP2-Common

Future Work

- Automatic Check pointing

- Additional back-end libraries
  - AVX Multi-cores, OpenCL

- Additional Diagnostics, Instrumentation and Performance Modelling
  - MPI + OpenMP achieved bandwidth figures
  - Performance modelling hybrid back-ends
  - Benchmarking power consumption

- Example/Prototype/Production Applications
  - Finite Element applications currently being developed using OP2
  - Rolls-Royce Hydra currently converted to OP2 at Imperial College London
  - AWE benchmarks ?

EPSRC Proposal – OP2 for Structured Meshes

- Increasing number of cores on a single chip
  - Need to keep cores fully utilized – memory bandwidth becoming a key bottleneck
  - Data movement on-chip is more power consuming than floating-point operations
  - On new architectures we need to optimise algorithms for data movements

"It's not about the FLOPS, it's about data movement"

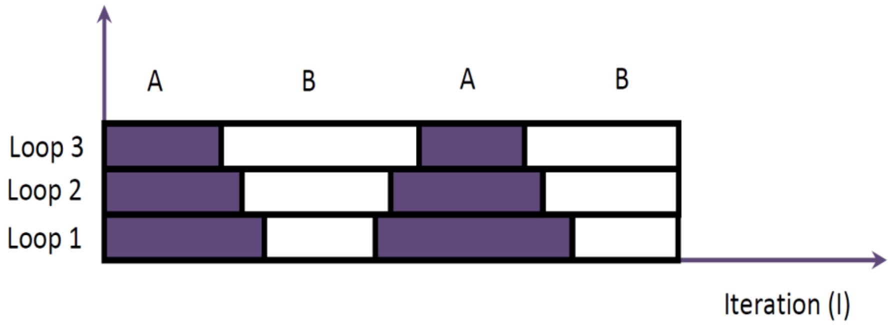
- Complex processor architectures roadmap
  - Need to achieve high productivity as well as high performance
  - Difficult to program emerging architectures and gain good performance
  - Code longevity - need to maintain near-optimal performance

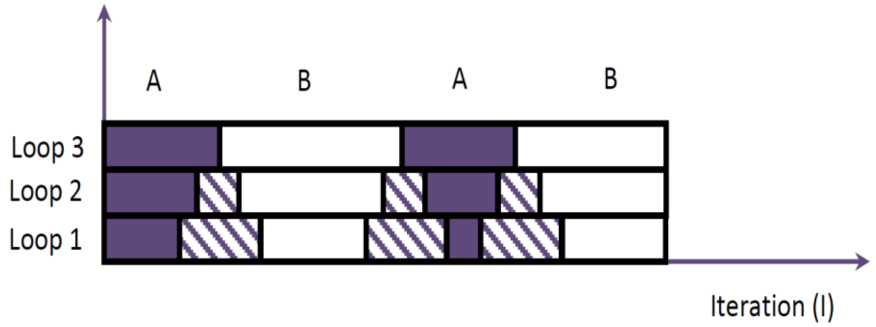
Data-Efficient Tiling - Simple Example

```
for(i=0; i<I; i++) res[i] = 0.0;               //loop 1

for(i=0; i<I-1; i++) {                         //loop 2
      flux = flux_function(q[i],q[i+1]);
      res[i] -= flux;
      res[i+1] += flux;
}

for(i=0; i<I; i++) q[i] += dt*res[i];          //loop 3
```





A towers computed in parallel, then *B* towers

Both *A* and *B* towers can be computed in parallel, with additional redundant computation

DATA-EFFICIENT TILING - SIMPLE EXAMPLE

```
for(i=0; i<I; i++) res[i] = 0.0;                    //loop 1

for(i=0; i<I-1; i++) {                              //loop 2
        flux = flux_function(q[i],q[i+1]);
        res[i] -= flux;
        res[i+1] += flux;
}

for(i=0; i<I; i++) q[i] += dt*res[i];               //loop 3
```

Standard Operation

Loop 1 : read/write res
Loop 2 : read q, read/write res
Loop 3 : read res, read/write q

Total : 8N transfers, where set size is N

Data-Efficient Tiling - Simple Example

```
for(i=0; i<I; i++) res[i] = 0.0;                 //loop 1

for(i=0; i<I-1; i++) {                           //loop 2
        flux = flux_function(q[i],q[i+1]);
        res[i] -= flux;
        res[i+1] += flux;
}

for(i=0; i<I; i++) q[i] += dt*res[i];            //loop 3
```

Tiling – non-redundant version

Loop 1 : read res, hold in cache
Loop 2 : read q, update res in cache
Loop 3 : update q, write out q/res when moving to next tower (forcing cache line to be displaced)

Total : 4N transfers – factor of 2x savings

Data-Efficient Tiling - Simple Example

```
for(i=0; i<I; i++) res[i] = 0.0;                //loop 1

for(i=0; i<I-1; i++) {                          //loop 2
        flux = flux_function(q[i],q[i+1]);
        res[i] -= flux;
        res[i+1] += flux;
}

for(i=0; i<I; i++) q[i] += dt*res[i];           //loop 3
```

Tiling – redundant version

Loop 1 : initialize in cache
Loop 2 : read q, update res in cache
Loop 3 : q, res in cache write out q

Total : 2N transfers – factor of 4x savings

** res data does not have to be stored back in main memory --- just hold a working set in cache **

This is much like the use of shared memory in the current GPU/OpenMP version

OXFORD e-Research CENTRE

EPSRC Proposal – OP2 for Structured Meshes

Objectives

- ❑Development of data-efficient tilling methods for PDEs based on structured and unstructured mesh based applications
- ❑ Extension of the OP2 API for solving unstructured mesh based applications to single-block structured mesh based applications.
- ❑ Implementation using lazy execution – evaluations are only performed as required
- ❑ Extension of the structured mesh API to multi-block structured mesh applications