Directive-Based Parallelization of the NIM

Mark Govett
Tom Henderson, Jacques Middlecoff,
Jim Rosinski, Paul Madden

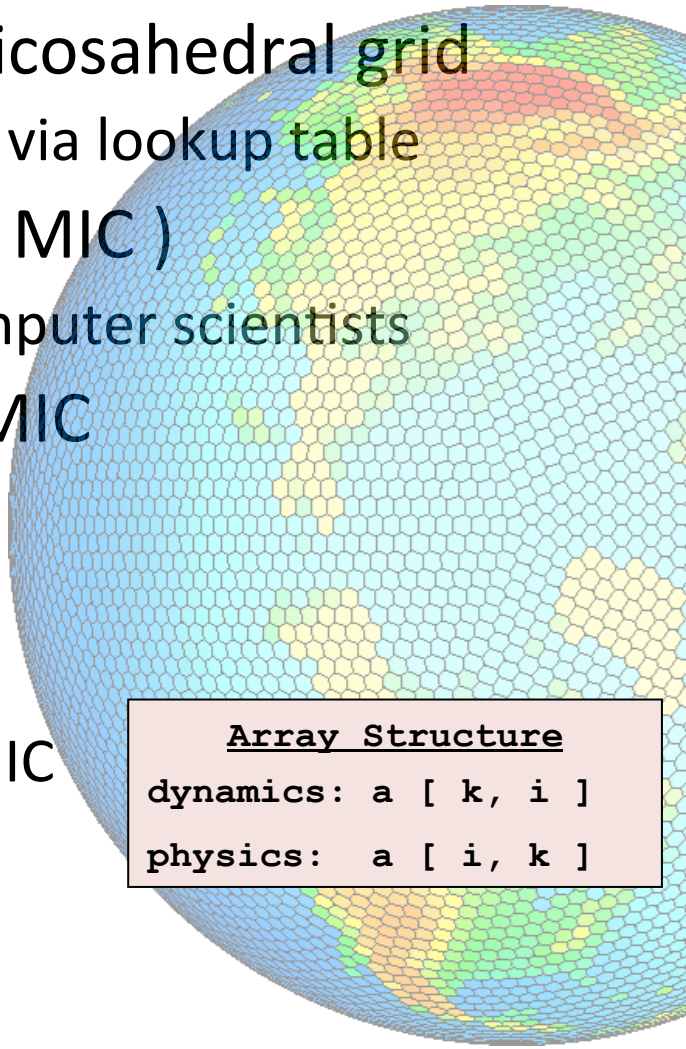
NOAA Earth System Research Laboratory

Outline

- Brief overview of NIM model
- GPU parallelization of NIM using F2C-ACC
  - Performance comparison to OpenACC Compilers
- Performance and Scaling
  - CPU, GPU, MIC
- Two talks to follow on MIC parallelization
  - Tom Henderson
    - report on MIC parallelization of WRF physics routines
  - Jim Rosinski
    - Parallelization and performance of NIM and FIM

Non-Hydrostatic Icosahedral (NIM)

- Global non-hydrostatic dynamical core
  - Designed to run at 3.5KM resolution or finer scales
- Uniform, global, hexagonal-based icosahedral grid
  - Single horizontal dimension, indexed via lookup table
- Co-designed in 2008 for GPU (and MIC)
  - Scientists, parallel programmers, computer scientists
- Single source code for CPU, GPU, MIC
  - Directives used for parallelization
    - OpenMP, OpenACC, F2C-ACC, SMS
- Good Performance & Scalability
  - Tested on up to 10000 GPUs, 1000 MIC
- Runs with GFS & WRF physics
  - Idealized, real-data tests in progress

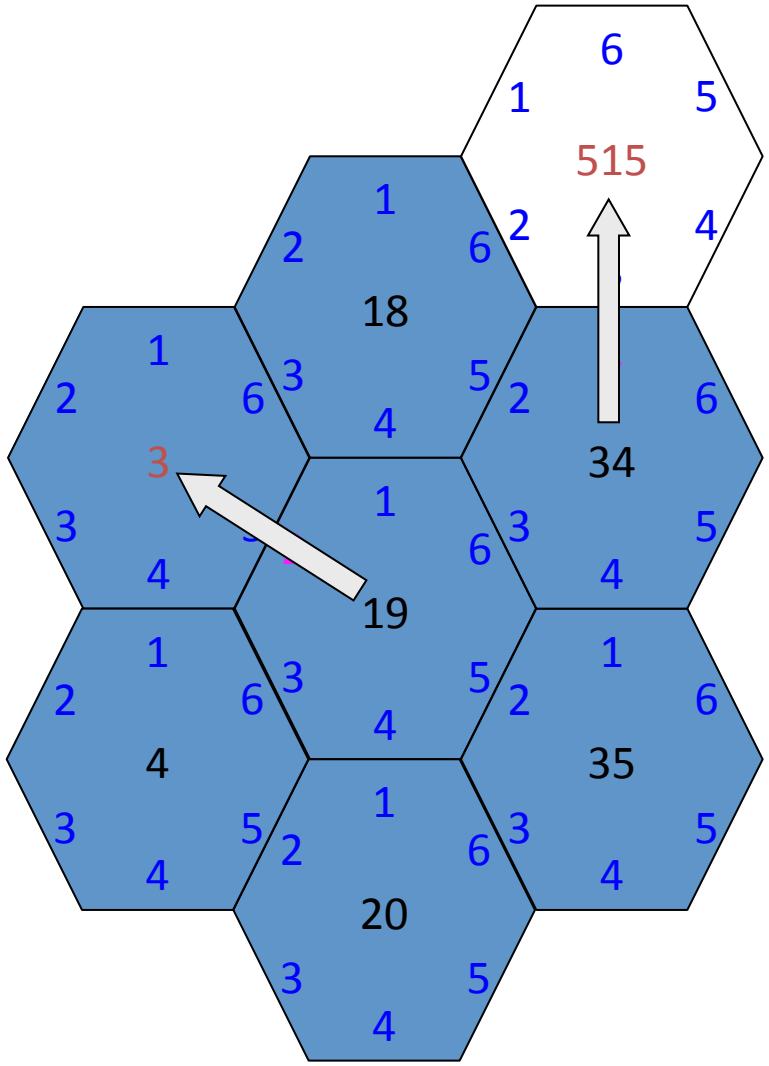


Array Structure

```
dynamics: a [ k, i ]
physics:  a [ i, k ]
```

NIM/FIM Icosahedral Grid

(MacDonald, Henderson, Middlecoff)



- Use a single horizontal index
- Store number of sides (5 or 6) in "nprox" array
  - nprox(34) = 6
- Store neighbor indices in "prox" array
  - prox(1,34) = 515
  - prox(2,19) = 3
- Array Organization
  - Vertical dimension innermost
    - Mitigates cost of grid cell lookup table
  - Indirect address costs ~1%
- Very compact code

Grid Decomposition

- Decomposition Strategy
  - Multiple of 10 MPI tasks is best
    - Maps directly to rhombuses
  - Eg. 60 MPI tasks means 10 per rhombus
- Grid layout within a MPI task
  - Neighbor points together
  - Square shape to minimize halo
- Graphic shows decomposition for 10 MPI tasks
  - Interior points in dark blue
  - Other colors are shared halo points

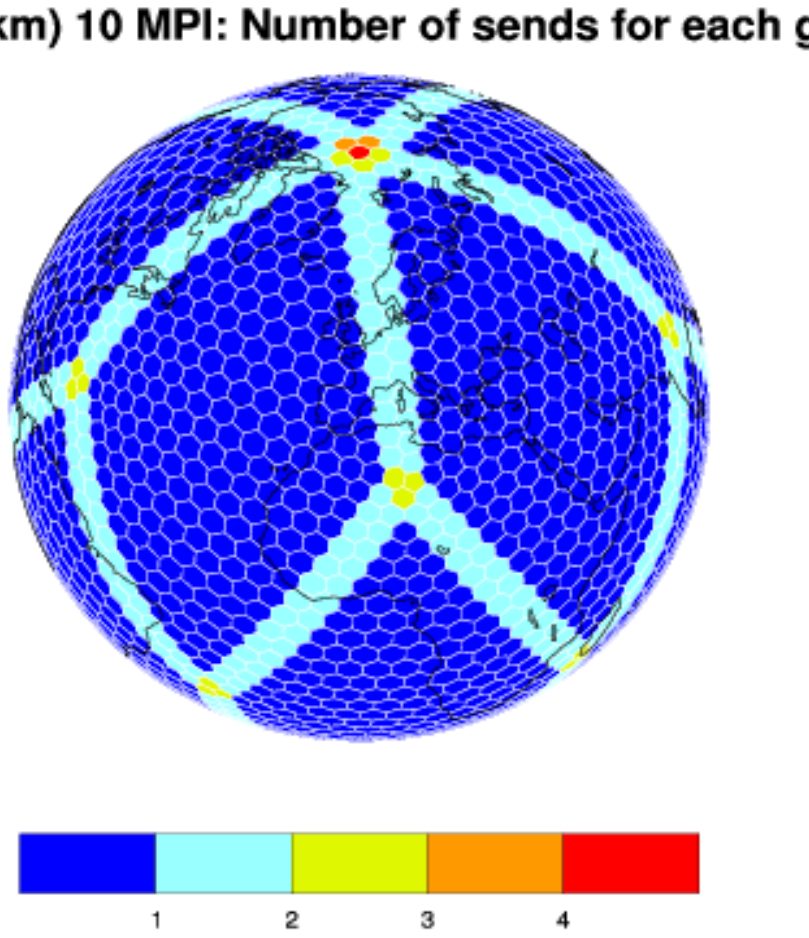


Image courtesy of Jim Rosinski

F2C-ACC Compiler

- Converts Fortran into CUDA
- Developed in 2008 before commercial compilers were available
- Limited Capabilities, Scope, Support
  - Mostly line for line code conversion with limited analysis
- Used for NIM, FIM and some WRF physics
- Performance optimizations **added to directives as options**
  - Define loop parallelism, GPU shared & local memory
- Used to evaluate OpenACC compilers (CAPS, PGI, Cray)
  - Capability: Can they support FIM, NIM, WRF?
  - Performance: Are they within ~10-20% of F2C-ACC?

Dynamics Code + F2C-ACC Directives

- Directives appear as Fortran comments
  - ACC$REGION defines an accelerator region
  - ACC$DO identifies parallelism
  - ACC$THREAD restricts parallelism to a single thread

```
!ACC$REGION(<96>,<10242>) BEGIN
!ACC$DO PARALLEL(1)
do ipn=ips,ipe          ! Loop over horizontal
!ACC$DO VECTOR(1)
do k=1,nz-1             ! Loop over vertical levels
    bedgvar(k,ipn,1) = ca4k(k)* u(k,ipn)+ca4p(k)* u(k+1,ipn)
    bedgvar(k,ipn,2) = ca4k(k)* v(k,ipn)+ca4p(k)* v(k+1,ipn)
  end do
!ACC$THREAD(nz-1) BEGIN
  bedgvar(nz,ipn,1)= ca4k(nz)* u(nz,ipn)+ca4p(nz)* u(nz,ipn)
  bedgvar(nz,ipn,2)= bedgvar(nz,ipn,2)=ca4k(nz)* v(nz,ipn)   &
    +ca4p(nz)* v(nz,ipn)
!ACC$THREAD END
end do
!ACC$REGION END
```

Code with F2C-ACC & OpenACC

```
!ACC$REGION(<96>,<10242>) BEGIN
!$acc parallel num_gangs(10242) vector_length(96)
!ACC$DO PARALLEL(1)
!$acc loop gang
do ipn=ips,ipe
!ACC$DO VECTOR(1)
!$acc loop vector
  do k=1,nz-1
    bedgvar(k,ipn,1) = ca4k(k)* u(k,ipn)+ca4p(k)* u(k+1,ipn)
    bedgvar(k,ipn,2) = ca4k(k)* v(k,ipn)+ca4p(k)* v(k+1,ipn)
  end do
!ACC$THREAD(nz-1) BEGIN
  bedgvar(nz,ipn,1)= ca4k(nz)* u(nz,ipn)+ca4p(nz)* u(nz,ipn)
  bedgvar(nz,ipn,2)= bedgvar(nz,ipn,2)=ca4k(nz)* v(nz,ipn)
!ACC$THREAD END
enddo
!$acc end parallel
!$acc end data
!ACC$REGION END
```

- OpenACC's kernel directive was also tried
  - Slower performance
  - User has less control over parallelization

OpenACC Compiler Evaluation (2014)

- Results shared with NVIDIA, PGI, Cray
  - Correctness: improving, bugs reported to vendors
  - Performance: Significantly slower than F2C-ACC

NIM Dynamics Runtimes in seconds, 100 time steps, single precision, single GPU

Routine	(% of CPU	F2C-ACC	PGI - OpenACC	Cray - OpenACC
Vdmints[1]	(38%)	7.10	18.40 (2.6)	14.61 (2.1)
Vdmintv	(15%)	3.59	7.19 (2.0)	5.50 (1.5)
Flux	(9%)	1.06	1.94 (1.8)	1.52 (1.4)
Diag	(8%)	0.81	2.00 (2.5)	0.89 (1.1)
Total	**(100%)**	**16.12**	**35.53 (2.2)**	**28.52 (1.8)**

[1] represents runtimes for 3 variants of the same routine

WRF Physics Runtimes in micro-seconds, 1 kernel invocation, double precision

Routine	(% total)	F2C-ACC	PGI - OpenACC	Cray - OpenACC
WSM3	(19%)	21.4	599.6 (28.0)	
PBL	(1%)	1.5	3.4 (2.3)	

OpenACC Compilers

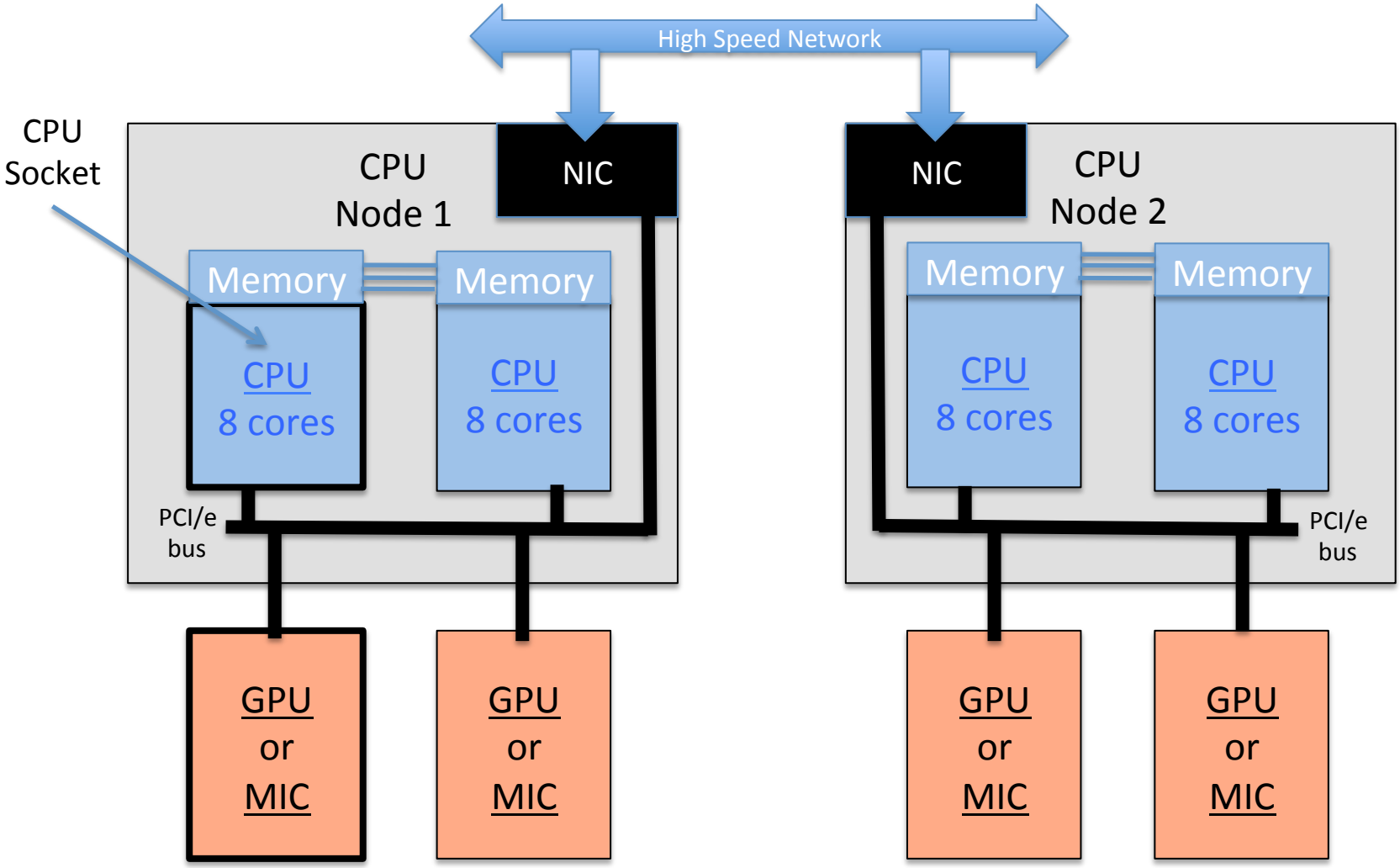
- Sharing results with vendors
  - Standalone tests to demonstrate bugs, performance issues
  - Vendors are responsive
- Concerns about standard
  - Prescriptive
    - More user control, via directive
  - Descriptive
    - Compiler does the analysis
    - Can ignore user specifications
    - Black box – limited ability to optimize
  - Making concerns known, suggesting improvements
    - I represent NOAA on the OpenACC committee

NIM Performance: CPU, GPU, MIC

- Ideal comparison
  - Identical source code
  - Same CPU chip, different accelerators
  - Multiple nodes linked using same interconnect
  - Same software stack
- This comparison
  - Identical source code
    - Changes that improve performance on one architecture cannot degrade on the others
  - Different CPU chips, same generation
  - Single node performance
    - not reliant on the interconnect
    - CPU, GPU, MIC only
    - Symmetric Mode: CPU + MIC, CPU + GPU

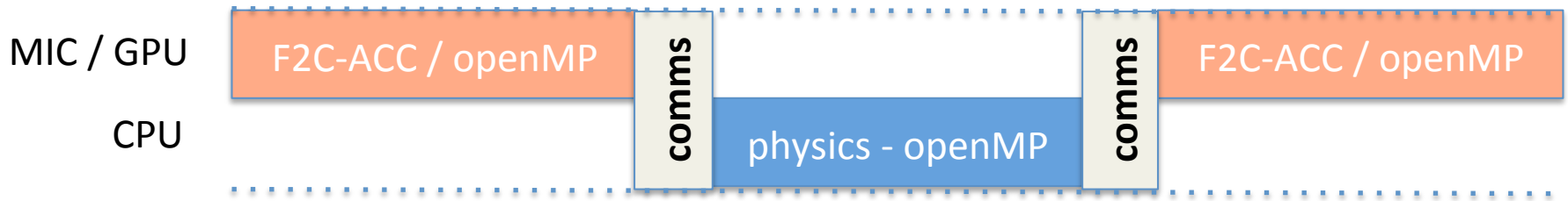
Node Configuration (2014)

- 2 nodes, 2 sockets, 2 accelerators

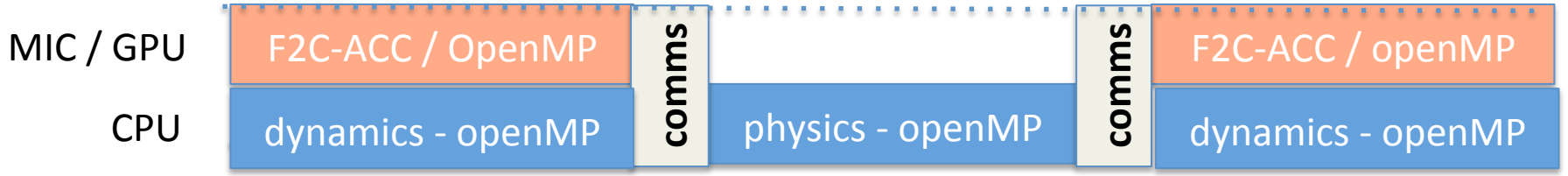


Symmetric Execution on the GPU & MIC

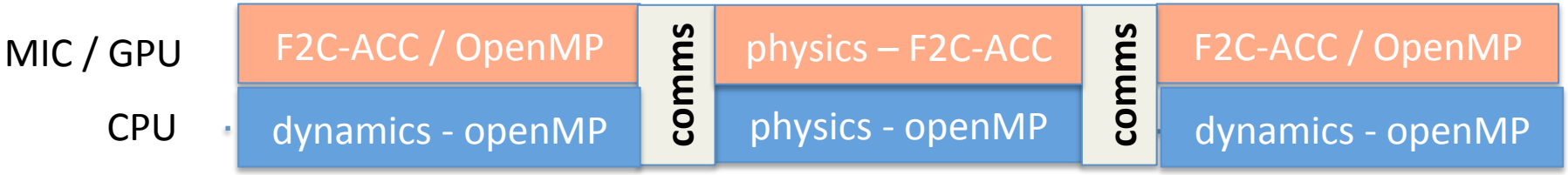
2012-2013: Dynamics on MIC or GPU + Physics on CPU



2013 / 2014: Symmetric Mode: divide the points between CPU and GPU or MIC



2015: Fully Symmetric: CPU + GPU, CPU + MIC



Single Node Performance: CPU, GPU,MIC

Parallelization and Performance

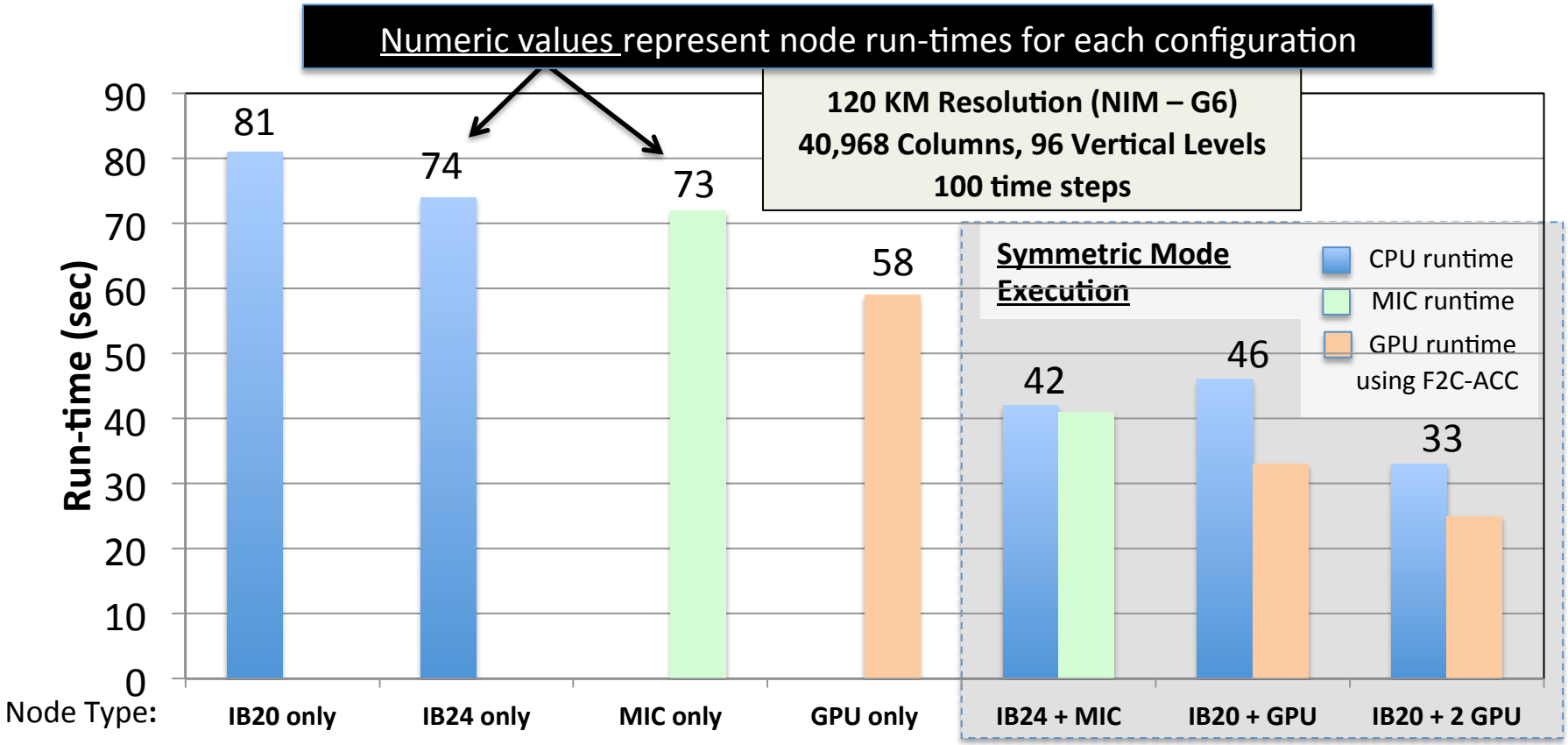
- Single source code (NIM rev 2724)
- Directive-based parallelization
  - **OpenMP** **CPU, MIC**
  - **F2C-ACC** **GPU**
  - **SMS** **MPI**
  - **OpenACC** **GPU**

System / Node configurations

- NVIDIA PSG Cluster
  - **IB20**: Intel IvyBridge, 20 cores, 3.0 GHz (Intel E5-2690 v2)
  - **GPU**: Kepler K40 2880 cores, 745 MHz, 12GB memory
- Intel Endeavor Cluster
  - **IB24**: Intel IvyBridge 24 cores, 2.70 GHz (Intel E5-2697 v2)
  - **MIC**: KNC 7120 61 cores, 1.238 GHz, 16 GB memory

http://www.esrl.noaa.gov/gsd/ab/ac/NIM-Performance.html

Single Node Performance: CPU, GPU, MIC



Parallelization and Performance

- Single source code (NIM rev 2724)
- Directive-based parallelization
  - **OpenMP** **CPU, MIC**
  - **F2C-ACC** **GPU**
  - **SMS** **MPI**
  - **OpenACC** **GPU**

System / Node configurations

- NVIDIA PSG Cluster
  - **IB20**: Intel IvyBridge, 20 cores, 3.0 GHz (Intel E5-2690 v2)
  - **GPU**: Kepler K40 2880 cores, 745 MHz, 12GB memory
- Intel Endeavor Cluster
  - **IB24**: Intel IvyBridge 24 cores, 2.70 GHz (Intel E5-2697 v2)
  - **MIC**: KNC 7120 61 cores, 1.238 GHz, 16 GB memory

http://www.esrl.noaa.gov/gsd/ab/ac/NIM-Performance.html

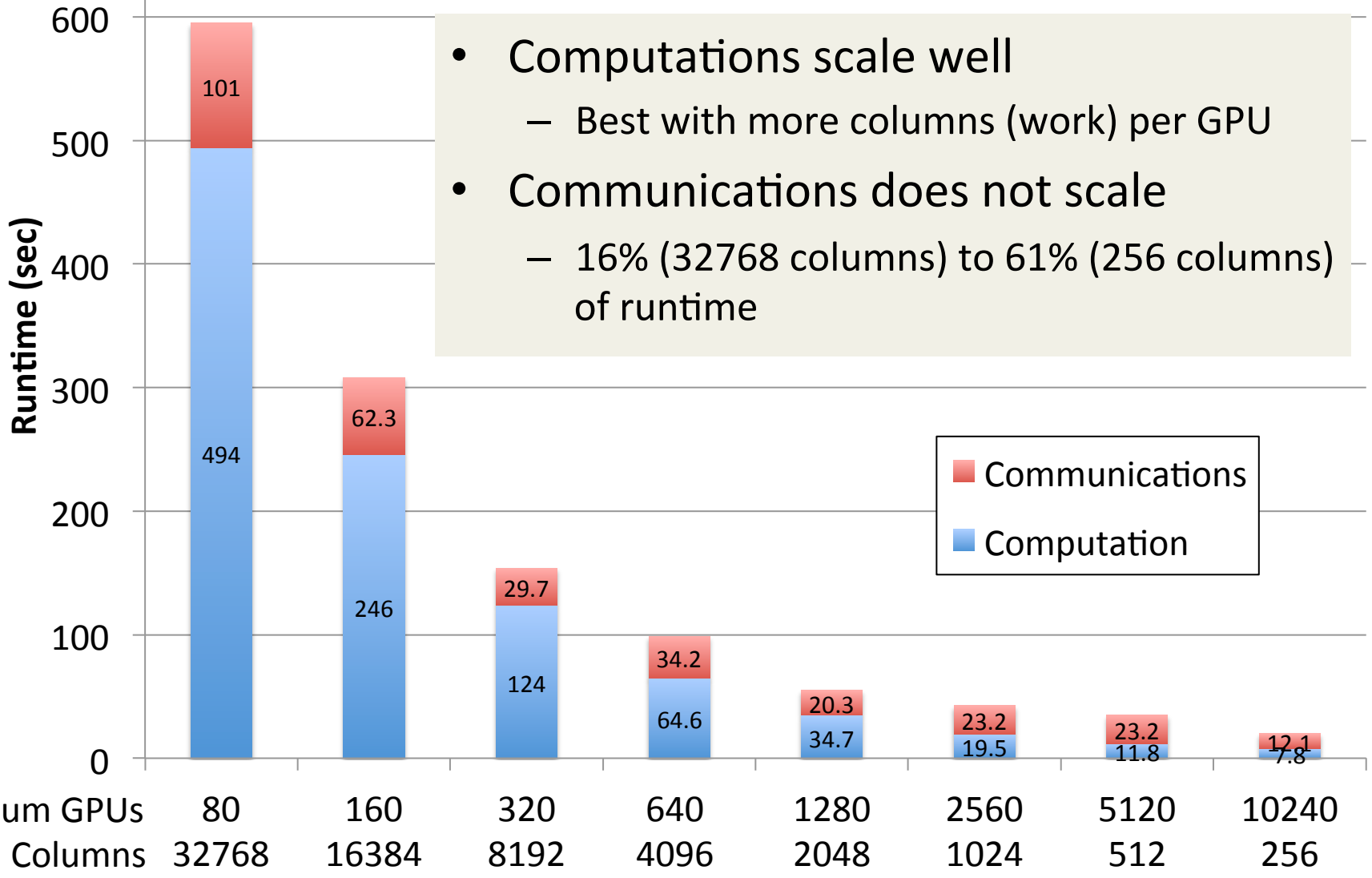
NIM: Scalability

Inter-process Communications

- Rely on Scalable Modeling System for inter-process communications
  - Directive-based, distributed memory parallelism (MPI)
  - Halo update modified to support GPU – to – GPU comms
    - 3 stages: Pack, MPI-EXCHANGE - Unpack
- Optimizations decrease communications overhead from over 50% of runtime to less than 17% on Titan
  - Move from individual pack & unpack kernels per variable to a single kernel
    - Template defining points to be exchanged set up at initialization
    - Gave a 3x speedup for pack and unpack runtimes
  - Use GPU mapped memory to improve GPU-CPU transfer times
    - Gave a 1.9x speedup in data copy
  - Overlap communications with computation

NIM Performance: Strong Scaling (2014)

15KM resolution, 1000 time steps, no I/O
NVIDIA Kepler K20x GPU



- Computations scale well
  - Best with more columns (work) per GPU
- Communications does not scale
  - 16% (32768 columns) to 61% (256 columns) of runtime

Communication Optimizations - 2014

- Eliminate pack and unpack completely (50% of communications)
  - Re-organize model grid points so data to be communicated is placed in message buffers
  - Pack and Unpack operations require half of the communications time
  - Indirect addressing makes it easy to reorganize data points that do not require changes to model code
- Further work to overlap communications with computation
  - Work with scientists to restructure calculations

Currently: halo points are stored in no particular order

Future: Columns will be re-organized so all halo points to a specific neighbor will be organized in separate pack and unpack message buffers, determined at model startup

Estimated Total Cost of Ownership

Based on 2014 Hardware

- Hardware Costs: based on list price
  - $5000 - dual socket, 20 cores IvyBridge
  - $3000 - Kepler K20x (Titan)
- Application Performance: Single node, CPU, CPU + GPU
- Annual Power Costs
  - TBD: power measurements for an application run are needed

Hardware	CPU	GPU	CPU + 1 GPU	CPU + 2 GPUs
Cost Factor ($$)	1 ($5000)	1.6 ($8000)	1.6 ($8000)	2.2 ($11000)
Perf Factor	1	1.4	1.8	2.5
cost benefit	0%	-20%	20%	30%
Perf Factor ++	1	1.4	2.5	3.3
cost-benefit ++	0%	-20%	90%	110%

++ Estimates based on optimizations to eliminate pack and unpack, per previous slide