Porting, validating, and optimizing NOAA/ESRL forecast models on Intel Xeon Phi (and Xeon)

October 28, 2014

Jim Rosinski
NOAA/ESRL

Outline

- Icosahedral grid and example communication pattern
- Current status of NIM and FIM dynamics performance
- Load balancing on heterogeneous hardware
- Lazy approach to code speedups
  - Performance improvements due to hardware upgrades
  - What if you just address threading and porting?
- Performance enhancements
- Communication issues
- Future directions

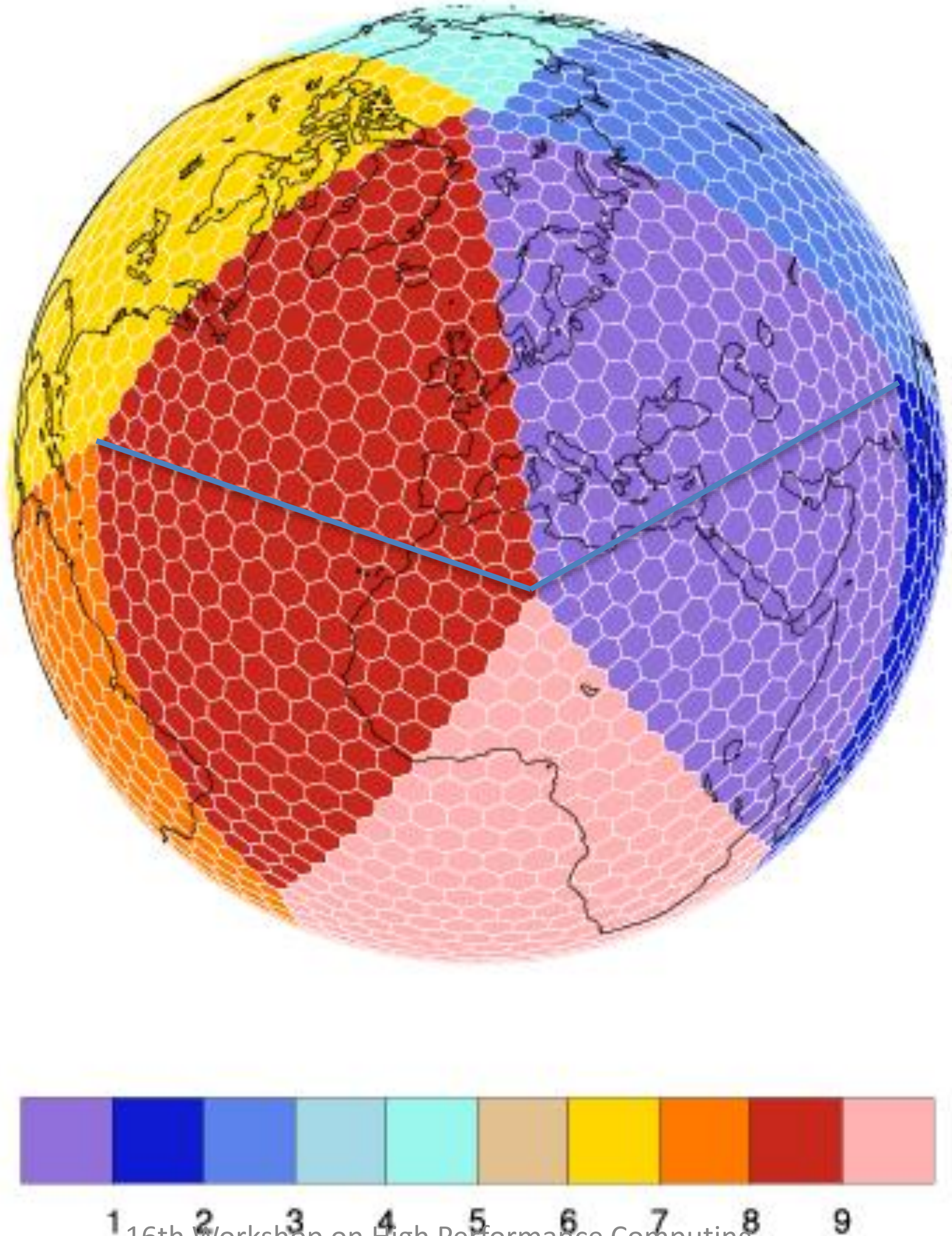
Contributors

- Tom Henderson (NOAA)
  - Physics porting and optimization
- Jacques Middlecoff (NOAA)
  - MPI optimizations
- Mike Greenfield (Intel)
  - Organizing Intel assistance to FIM and NIM efforts
- Ruchira Sasanka (Intel)
  - Optimizing MPI communication
  - General optimization efforts
- Ashish Jha (Intel)
  - Alignment optimizations
  - Compiler flag suggestions

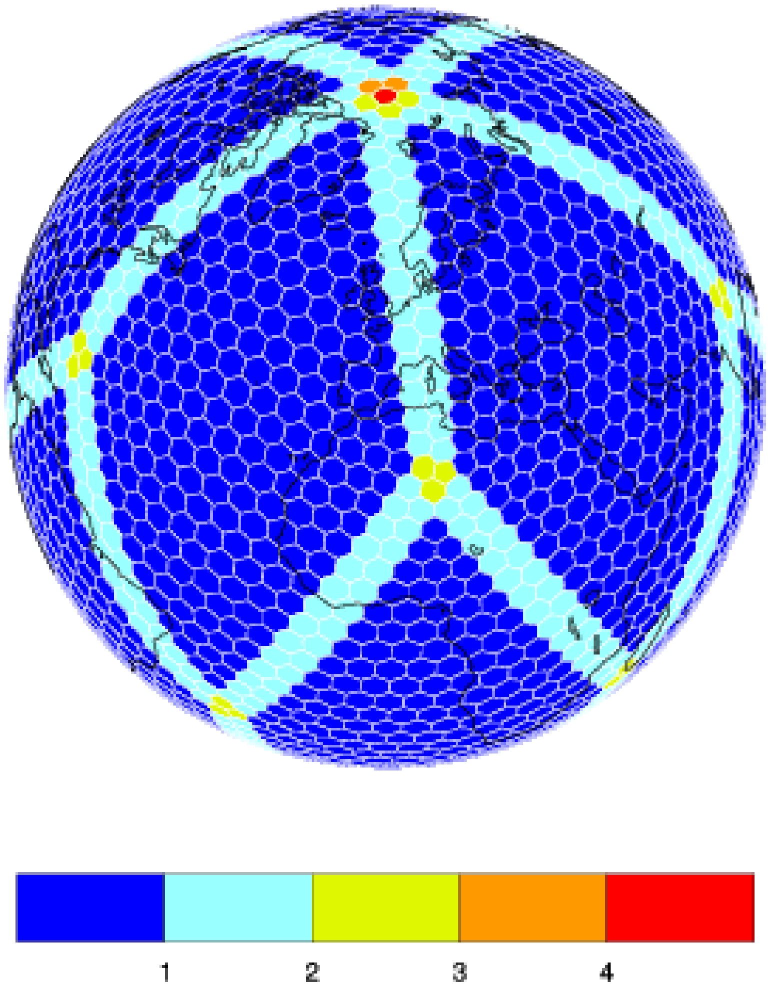
What is NIM?

- Non-hydrostatic Icosahedral Model
- Weather forecast model (up to 10 days)
  - Designed for very high resolution (< 10 km)
  - Improved forecast performance over terrain
- Software:
  - Optimize performance on current scientific target platform (Xeon-based)
  - Maintain code base in single-source
  - Port to multiple platforms including MIC, GPU
  - Validate model solutions on all ported platforms

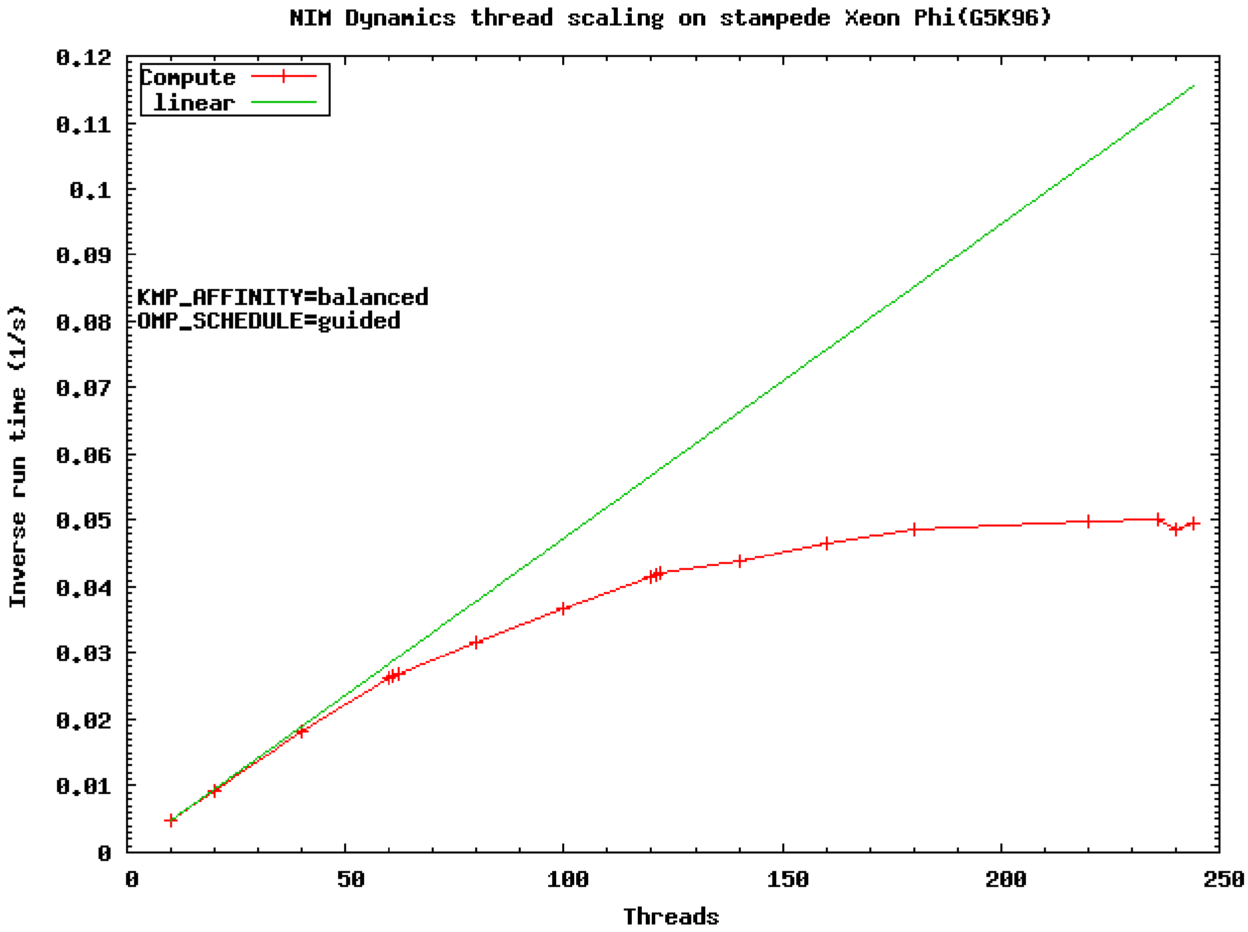
NIM G4 (446 km) point allocation on 10 MPI tasks



NIM G4 (446 km) 10 MPI: Number of sends for each grid point



NIM thread scaling on MIC



NIM single-node performance on various hardware (100 km)

Node configuration	MPI tasks	Runtime for 100 time steps	Hardware specs (system=Intel endeavor)
Host-only IVB	2	67.1 sec	Xeon E5-2697v2, 2.7 GHz, 24 cores
Host-only HSW-EP	2	57.4 sec	Xeon E5-2697v3, 2.6 GHz, 28 cores
MIC-only	2	67.4 sec	1.23 GHz, 61 cores
Symmetric IVB+KNC	5+5	39.6 sec	See above

NIM speedups on CPU due to hardware improvements

Architecture	CPU specs	Memory specs	NIM dynamics time on 1 node	% speedup vs. SNB
SandyBridge (stampede)	16 cores Intel Xeon E5-2680@2.7GHz	ddr3 1600 Mhz	92.1 sec	0%
IvyBridge (endeavor)	24 cores Intel Xeon E5-2697v2@2.7GHz	ddr3 1600 Mhz	67.1 sec	27%
Haswell-EP (endeavor)	28 cores Intel Xeon E5-2697v3@2.6Ghz	ddr4 2100 Mhz	57.4 sec	38%

FIM model dynamics performance (200 km, 64 levels, 10000 grid points, 1 node)

Routine	SNB time (s)	MIC time (s)
main_loop	19.658	36.600
dynamics	16.072	32.104
hybgen	4.532	5.818
edgvar1-2	4.459	10.784
cnuity	2.033	2.079
trcadv	3.689	3.803
cpl_run	3.558	4.183

Claim based on FIM results

- "Reasonable" performance on MIC can be expected (compared to host) if code meets these criteria:
  - Highly parallel (e.g. > 99%)
  - Enough thread contexts can be employed to keep all cores busy
  - Threaded loops contain enough work to amortize thread start-up and synchronization
  - Good inner loop vectorization

Symmetric mode load balancing

name	ncalls	nranks	mean_time	std_dev	wallmax	(rank)	wallmin	(rank)
Diag	1002	2	3.314	1.979	4.713	(0)	1.914	(1)
MainLoop	2	2	50.294	0.176	50.419	(0)	50.170	(1)
ZeroTendencies	200	2	0.093	0.022	0.108	(0)	0.077	(1)
SaveFlux	200	2	0.149	0.052	0.186	(0)	0.112	(1)
Dyntnc	800	2	38.277	1.794	39.546	(1)	37.009	(0)
RHStendencies	800	2	0.419	0.139	0.518	(0)	0.321	(1)
Vdm	800	2	23.368	2.984	25.478	(0)	21.258	(1)
Vdmintv	800	2	6.462	0.282	6.661	(0)	6.262	(1)
Vdmints0	800	2	5.567	0.693	6.057	(0)	5.076	(1)
Vdmints3	800	2	8.506	1.037	9.240	(0)	7.773	(1)
vdmfinish	800	2	2.820	0.986	3.517	(0)	2.122	(1)
Vdn	800	2	1.806	0.224	1.965	(0)	1.648	(1)
Flux	800	2	3.676	0.105	3.750	(0)	3.601	(1)
Force	800	2	1.650	0.071	1.700	(0)	1.600	(1)
RKdiff	800	2	1.411	0.197	1.551	(0)	1.271	(1)
TimeDiff	800	2	0.706	0.237	0.873	(0)	0.538	(1)
Sponge	800	2	0.365	0.088	0.427	(0)	0.303	(1)
pre_trisol	200	2	0.139	0.019	0.153	(1)	0.126	(0)
Trisol	200	2	0.416	0.114	0.497	(0)	0.336	(1)
post_trisol	200	2	0.076	0.004	0.079	(0)	0.073	(1)
Vdmints	200	2	3.499	0.303	3.714	(0)	3.285	(1)
Pstadv	200	2	0.792	0.029	0.813	(1)	0.772	(0)

Symmetric mode load balancing (cont'd)

name	ncalls	nranks	mean_time	std_dev	wallmax	(rank	)	wallmin	(rank	)
Diag	5010	10	2.494	1.070	3.766	(	0)	1.616	(	9)
MainLoop	10	10	47.567	0.111	47.697	(	2)	47.480	(	8)
ZeroTendencies	1000	10	0.071	0.009	0.094	(	0)	0.063	(	1)
SaveFlux	1000	10	0.103	0.022	0.135	(	2)	0.078	(	4)
Dyntnc	4000	10	37.124	0.889	37.863	(	7)	36.022	(	0)
RHStendencies	4000	10	0.316	0.042	0.357	(	1)	0.243	(	9)
Vdm	4000	10	22.724	2.322	24.622	(	9)	19.964	(	3)
Vdmintv	4000	10	6.513	1.062	7.361	(	8)	5.253	(	3)
Vdmints0	4000	10	5.506	0.637	6.026	(	9)	4.749	(	3)
Vdmints3	4000	10	8.442	1.002	9.389	(	9)	7.264	(	1)
vdmfinish	4000	10	2.243	0.394	2.726	(	0)	1.815	(	9)
Vdn	4000	10	1.623	0.101	1.710	(	8)	1.451	(	3)
Flux	4000	10	3.629	0.509	4.053	(	4)	3.020	(	0)
Force	4000	10	1.487	0.147	1.639	(	9)	1.299	(	2)
RKdiff	4000	10	1.074	0.108	1.202	(	1)	0.913	(	6)
TimeDiff	4000	10	0.583	0.060	0.665	(	0)	0.520	(	6)
Sponge	4000	10	0.284	0.007	0.299	(	0)	0.274	(	1)
pre_trisol	1000	10	0.076	0.009	0.088	(	0)	0.064	(	9)
Trisol	1000	10	0.392	0.006	0.400	(	2)	0.385	(	9)
post_trisol	1000	10	0.057	0.009	0.065	(	7)	0.045	(	3)
Vdmints	1000	10	3.493	0.482	3.900	(	9)	2.918	(	3)
Pstadv	1000	10	0.803	0.162	0.944	(	4)	0.611	(	1)

Two simple but important mods affecting host and KNC performance

Runtime PRIOR to code mods (sec)		
Routine	SNB	KNC
diag	6.2	1.5
trisol	0.5	1.5

Runtime AFTER code mods (sec)		
Routine	SNB	KNC
diag	4.7	1.4
trisol	0.5	0.4

Host compiler issue (diag.F90)

- Vector loop gets fused with scalar loop:

```
! Line 93: This loop cannot vectorize due to a dependency
 do k=nz-1,0,-1
   p(k,ipn) = p(k+1,ipn) + pdel(k+1)
 end do

! Line 111: This loop can easily vectorize
 do k=1,nz
   term(k) = rd*tr(k,ipn)*1.e-5_rt
 end do

diag.f90(93): (col. 5) remark: loop was not vectorized: existence of vector dependence
Fused Loops: ( 93 111 )
```

- Solution: add “nofusion” directives to unvectorizable loops:

```
! Line 93: This loop cannot vectorize due to a dependency
!DIR$ NOFUSION
 do k=nz-1,0,-1
   p(k,ipn) = p(k+1,ipn) + pdel(k+1)
 end do
```

MIC compiler issue (trisol.F90)

- Vector loop gets fused with scalar loop:

```
! Line 97: This loop can vectorize even though it has many computations
  do k=1,nz-1
    kp1 = k+1
    km1 = k-1
    thkp1 = .5_rt*( bedgvar(kp1,ipn,6)+bedgvar(k,ipn,6))
    thkp  = .5_rt*( bedgvar(km1,ipn,6)+bedgvar(k,ipn,6))
... Lots more vectorizable code
  end do
...
! Line 139: This loop cannot vectorize because there is a dependency (w1d)
 do k=2,nz
    alpha = 1._rt/(bbb(k)-aaa(k)*gama(k-1))
    gama(k) = ccc(k)*alpha
    w1d(k) = (rrr(k)-aaa(k)*w1d(k-1))*alpha
 end do
```

- ifort -opt-report-phase=hlo -vec-report6 says:

```
fused Loops: ( 97 139 )
fused Loops: ( 84 97 )
trisol.f90(84): (col. 3) remark: loop was not vectorized: existence of vector dependence
```

MIC compiler issue (trisol.F90 cont'd)

- Solution: add "nofusion" directive to unvectorizable loop:

```
! Line 139: Disallow loop fusion of unvectorizable loop
!DIR$ NOFUSION
 do k=2,nz
    alpha = 1._rt/(bbb(k)-aaa(k)*gama(k-1))
    gama(k) = ccc(k)*alpha
    w1d(k) = (rrr(k)-aaa(k)*w1d(k-1))*alpha
 end do

trisol.f90(84): (col. 3) remark: FUSED LOOP WAS VECTORIZED
```

Improving thread scaling

- Original code (packs/unpacks MPI messages around sends/recvs):

```
do varNumber = 1,IVRBL                         ! Number of variables (typically around 4)
  var => exchPtr(varNumber)%varptr
  do n = 1,NumSendsOrRecvs                     ! Number of neighbors (typically 6-7)
!$OMP PARALLEL DO PRIVATE (jindirect, offset, i)
    do j = 1,numberToPackOrUnpk(n,varNumber) ! Number of grid points (typically
O(1000))
      jindirect = varIndexes(j,n,varNumber)
      offset    = bufIndexes(j,n,varNumber)
      if(pack) then !Pack the buffer
        do i = js(varNumber),je(varNumber)
          buffer(i+offset,n) = var(i,jindirect)
        enddo
      else ! Unpack the buffer
        do i = js(varNumber),je(varNumber)
          var(i,jindirect) = buffer(i+offset,n)
        enddo
      endif
    enddo
  enddo
enddo
```

Improving thread scaling (cont'd)

- Modified code code: threads don't synchronize until outer loop completes:

```
!$OMP PARALLEL PRIVATE (varnumber, var, n, j, jindirect, offset, i)
do varNumber = 1,IVRBL                       ! Number of variables (typically around 4)
  var => exchPtr(varNumber)%varptr
  do n = 1,NumSendsOrRecvs                   ! Number of neighbors (typically 6-7)
!$OMP DO
    do j = 1,numberToPackOrUnpk(n,varNumber) ! Number of grid points (typically
O(1000))
      jindirect = varIndexes(j,n,varNumber)
      offset    = bufIndexes(j,n,varNumber)
      if (pack) then ! Pack the send buffer from user space
        do i = js(varNumber),je(varNumber)
          buffer(i+offset,n) = var(i,jindirect)
        enddo
      else           ! Unpack the recv buffer into user space
        do i = js(varNumber),je(varNumber)
          var(i,jindirect) = buffer(i+offset,n)
        enddo
      endif
    enddo
!$OMP END DO NOWAIT
  enddo
enddo
!$OMP END PARALLEL
```

Compile-time vs. run-time array sizing and loop bounds specification (SNB)

```
module resolution
#ifdef RUNTIME
  integer :: nz                         ! Set at run-time
#else
  integer, parameter :: nz = NZ         ! cpp sets at compile-time
#endif
end module resolution
```

NIM performance compile-time vs. run-time array sizing and loop bounds specification (SNB)

Routine	Run-time (sec)	Compile-time (sec)	% speedup
Total	54.102	45.002	16.8%
vdmints3	10.271	7.576	26.2%
vdmints0	5.987	5.547	7.3%
vdmintv	6.663	6.186	7.2%

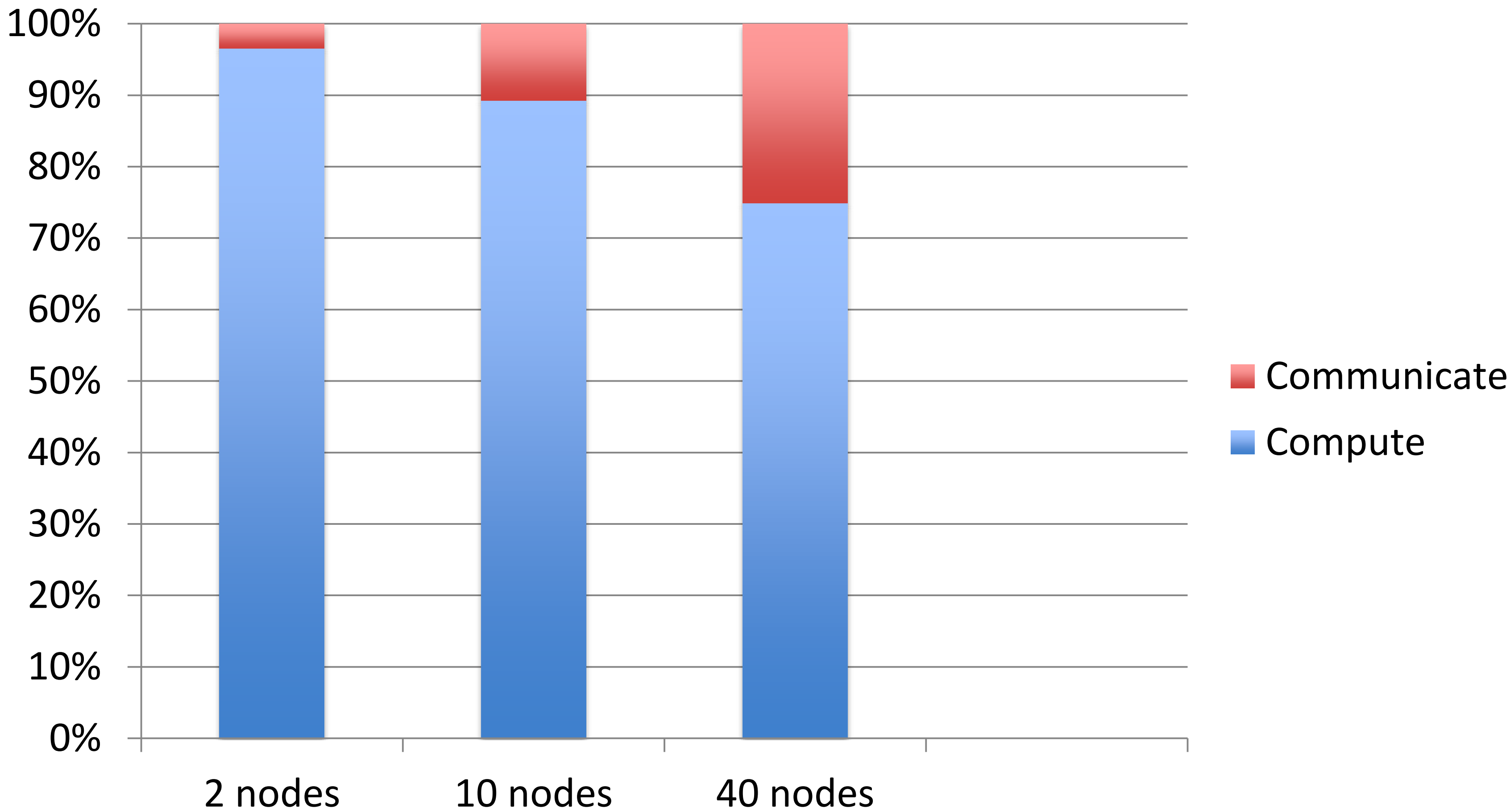
NIM performance compile-time vs. run-time array sizing and loop bounds specification (MIC)

Routine	Run-time (sec)	Compile-time (sec)	% speedup
Total	44.681	39.115	12.5%
vdmints3	7.975	6.389	19.9%
vdmints0	5.120	4.303	16.0%
vdmintv	6.432	5.257	18.2%

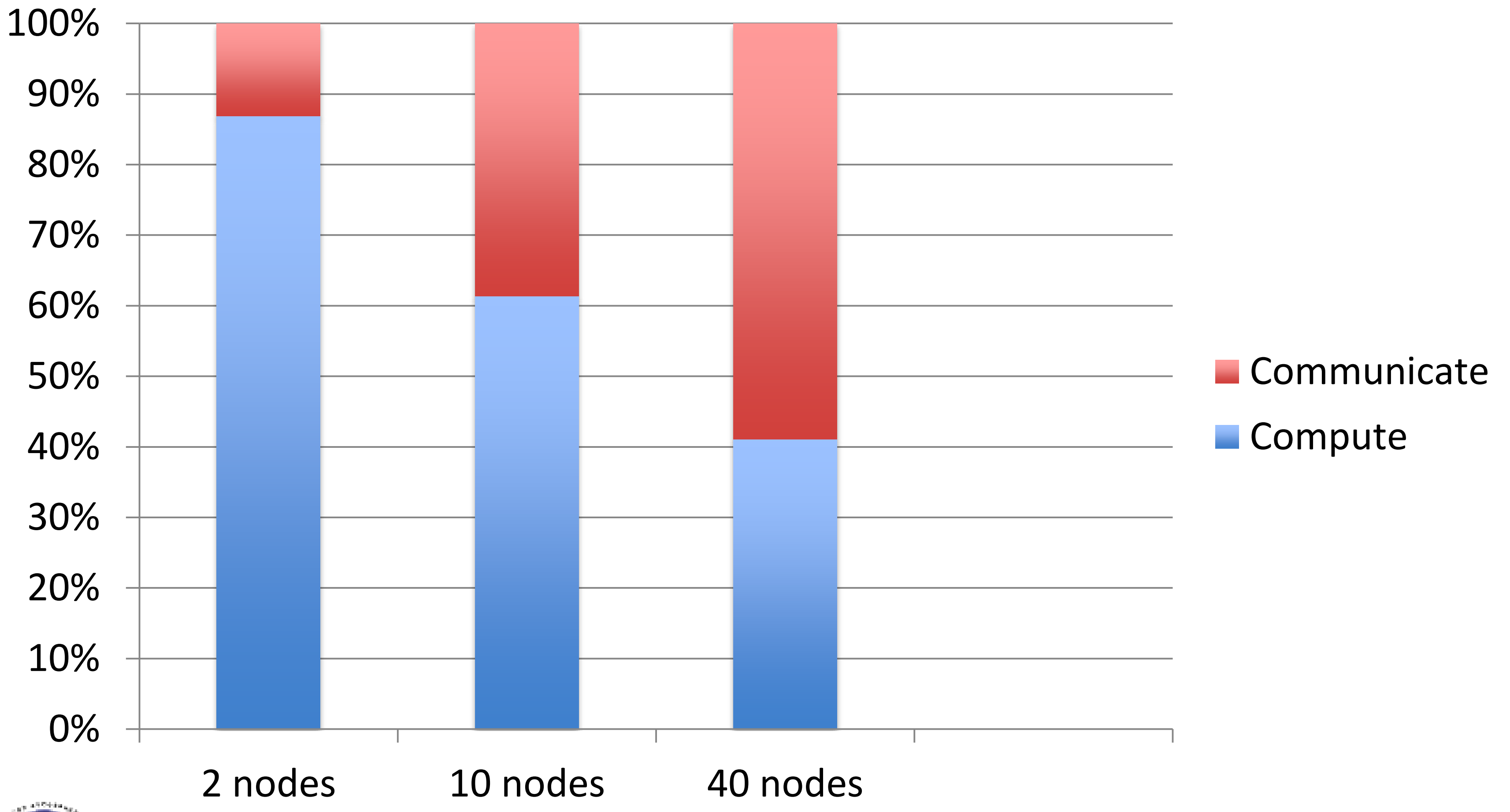
Validation

- NIM dynamics can be made to produce bitwise-identical answers Xeon vs. MIC if canonical transcendental functions are used.
  - No reductions which feed back into model calculations (vector, OMP, or MPI)
- Software constraint: NIM must produce bitwise identical answers across varying MPI task counts
  - -fp-model precise required on host compilation
- Intel provided us prototype math libraries for Xeon and Phi that produce bitwise identical results for transcendental functions (e.g. exp, log, pow, sin, cos). The library is not performance optimized, but allows us to unambiguously validate the port to Phi.

G6K96 (100 km) relative cost compute vs. communicate on SNB



G6K96 (100 km) relative cost compute vs. communicate on MIC



Where Next?

- Further enhance communication
  - Why is pack/unpack performance still slower on MIC vs. SNB?
  - Try replacing MPI_Isend with MPI_Irsend since message sizes are large?
  - Rewrite MPI calls to avoid pack/unpack (exploratory work by Jacques Middlecoff)
- GFS physics in FIM

Summary

- Full NIM dynamical core ported and validated on Xeon Phi using symmetric mode
  - Scientists are working on topography
- FIM dynamical core ported to MIC. Answers "probably" correct (validation pending)
- Single-source for CPU, Phi, GPU
- Dynamics running reasonably well on Phi (NIM performance matches IVB node)
- Inter-process comms are biggest performance challenge on MIC, GPU

Backup slides

Pack/Unpack timing results (1 rank per device, total runtime around 50 sec)

Architecture	OLD Pack/Unpack time (s)	NEW Pack/Unpack time(s)	% speedup
SNB	0.39	0.35	10%
MIC	0.71	0.50	29%

Primary changes to NIM dynamics in 2014

- Truly single-source for CPU/KNC/GPU
  - Constraint: mods to 1 architecture cannot degrade performance on another
  - Very few architecture-specific ifdefs
- Ability to run in real*8 mode (Tom Henderson)
- Compute rather than read in giant arrays on initialization
  - Allows high resolution runs
- Special transcendental libraries allow bitwise-exact results host vs. KNC (thanks to Intel math libraries team)
- Changes to SMS library improve communication performance (Jacques Middlecoff)
  - Better threading helps CPU/KNC
  - Fewer kernel calls helps GPU
- Mods to diag.F90, trisol.F90 improve performance on CPU/GPU/KNC

Multi-core performance (32-bit)

- Peak on SNB: 16 cores * 8 flops/clock/core * 2 vector instructions * 2.6 GHz = 665.6 Gflops/s
- NIM observed on SNB: 1.63e12 flops / 22.423 sec / 665.6e9 peak flops= 11% of peak

- Peak on KNC: 61 cores * 16 flops/clock/core * 1.238 GHz = 2.416 Tflops/s
- NIM observed on KNC: 1.63e12 flops / 22.254 sec / 2.416e12 peak flops/s = 3% of peak

- NIM observed on K20X GPU: 1.63e12 flops / 17.924 sec / 3.95e12 peak flops/s = 2% of peak

Dynamics

- Solves equations of motion for large-scale flow
- Little dependence in vertical
  - Fortran array organization is (vertical,horizontal)
- Computational cost grows as the cube of the inverse horizontal grid spacing

General looping structure in NIM dynamics

```
!ACC$DO PARALLEL(1)
!$OMP PARALLEL DO PRIVATE(k,isn,ipp,isp,fx1,fx2,fx5,fx,kp1,vnkm1,vnk,upfx1,upfx2,upfx3)
do ipn=ips,ipe
!ACC$DO VECTOR(1)
  do k=1,nz
    fx1(k) = 0.
    fx2(k) = 0.
    fx5(k) = 0.
  end do
  do isn=1,nprox(ipn)      ! loop thru edges getting fluxes
    ipp=prox( isn,ipn)
    isp=proxs(isn,ipn)
!ACC$DO VECTOR(1)
    do k=1,nz
      tefr (k,isn,ipn) =  .5*(vdns(k,isp,ipp)+abs(vdns(k,isp,ipp)))  &
                        - .5*(vdns(k,isn,ipn)+abs(vdns(k,isn,ipn)))
      fx1(k)  = fx1(k)+.5*(vdns(k,isn,ipn)+abs(vdns(k,isn,ipn)))*sedgvar(k,isn,ipn,1)*sa(k,isn,ipn) &
                     -.5*(vdns(k,isp,ipp)+abs(vdns(k,isp,ipp)))*sedgvar(k,isp,ipp,1)*sa(k,isp,ipp)
      fx2(k)  = fx2(k)+.5*(vdns(k,isn,ipn)+abs(vdns(k,isn,ipn)))*sedgvar(k,isn,ipn,2)*sa(k,isn,ipn) &
                     -.5*(vdns(k,isp,ipp)+abs(vdns(k,isp,ipp)))*sedgvar(k,isp,ipp,2)*sa(k,isp,ipp)
      fx5(k)  = fx5(k)+.5*(vdns(k,isn,ipn)+abs(vdns(k,isn,ipn)))*sedgvar(k,isn,ipn,5)*sa(k,isn,ipn) &
                     -.5*(vdns(k,isp,ipp)+abs(vdns(k,isp,ipp)))*sedgvar(k,isp,ipp,5)*sa(k,isp,ipp)
    end do
  end do ! end of loop through edges getting fluxes
```

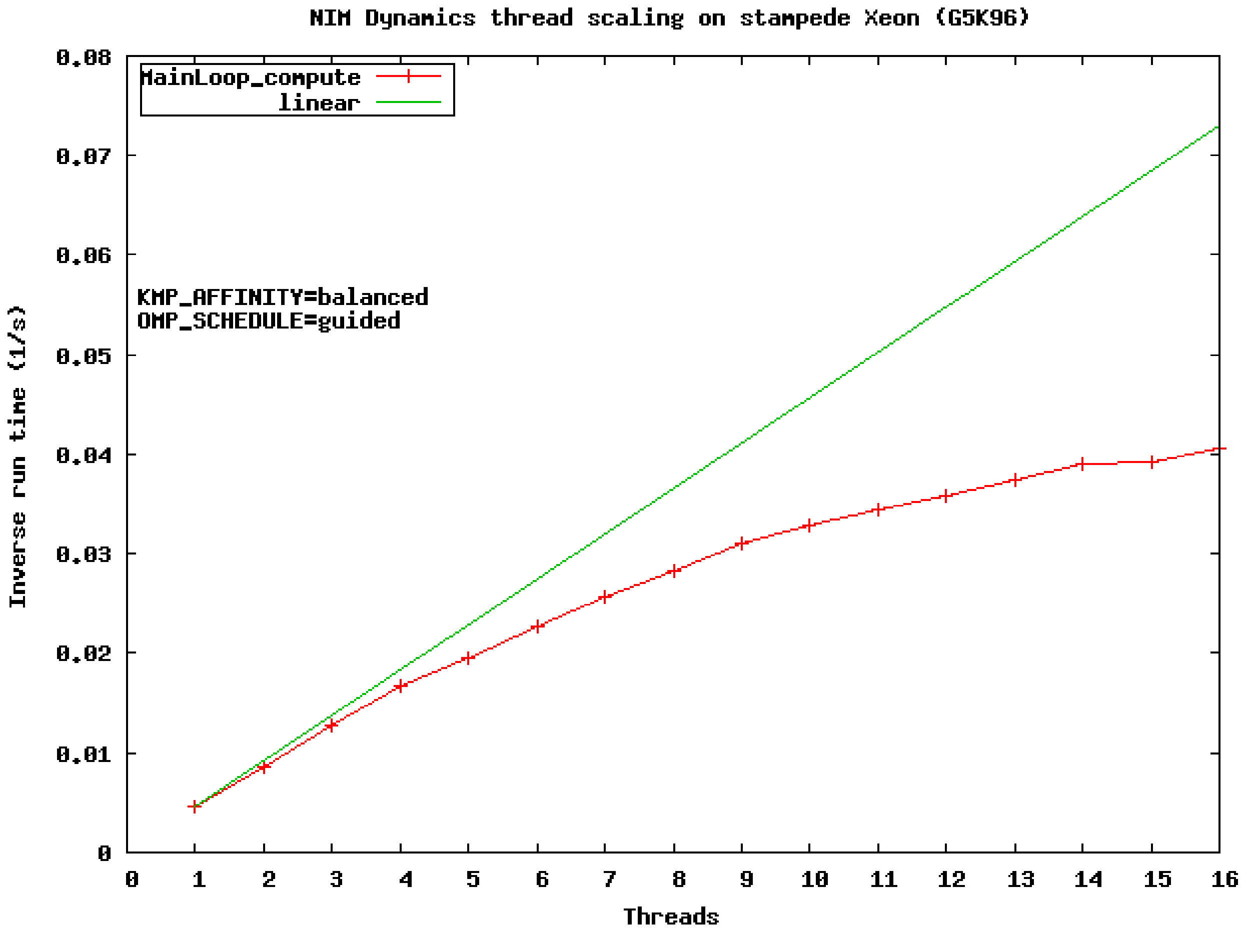
NIM Porting Methodology for symmetric mode on Phi

- Hybrid OpenMP/MPI
  - NIM was already parallelized for MPI => add OMP
- Modify compiler flags
  - Add –mmic for MIC build
  - Remove –fpe0
- Build a second executable as normal for Xeon
  - Required libraries (SMS, GPTL) also needed separate compilations
- stampede: ibrun –c <host_path> -m <mic_path>

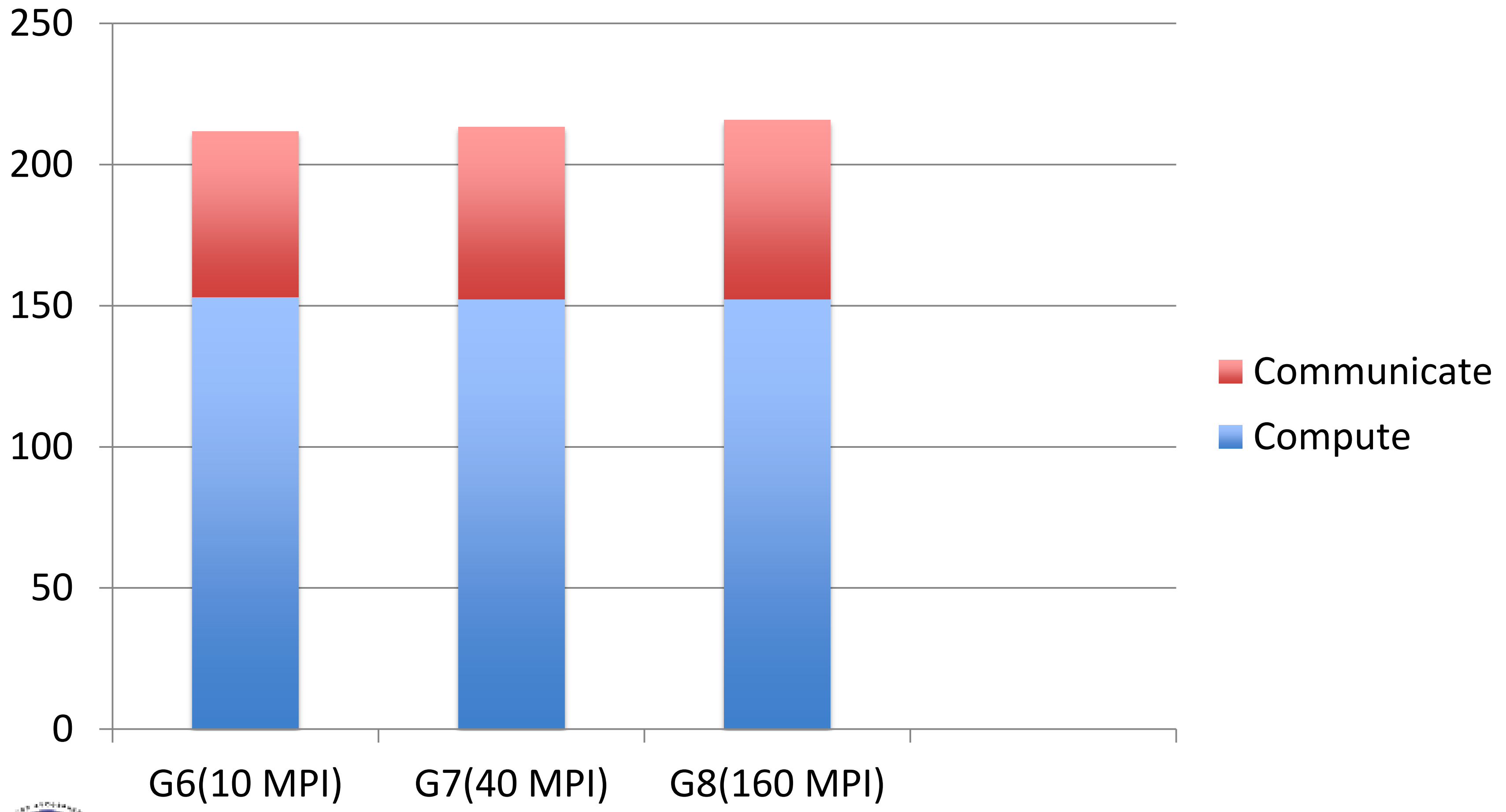
Optimizations for Phi

- Ensure that inner loops vectorize
- Ensure good thread scaling
- Alignment: -align array64byte
  - Add !DIR$ ASSUME_ALIGNED where appropriate
- Try other flags for optimization
  - –opt-streaming-stores always (memory bound codes)
- Vary from default OMP settings
  - OMP_SCHEDULE=guided (vs. static or dynamic,…)
  - KMP_AFFINITY=balanced (vs. scatter or compact)
- Replace divides with multiply by reciprocal

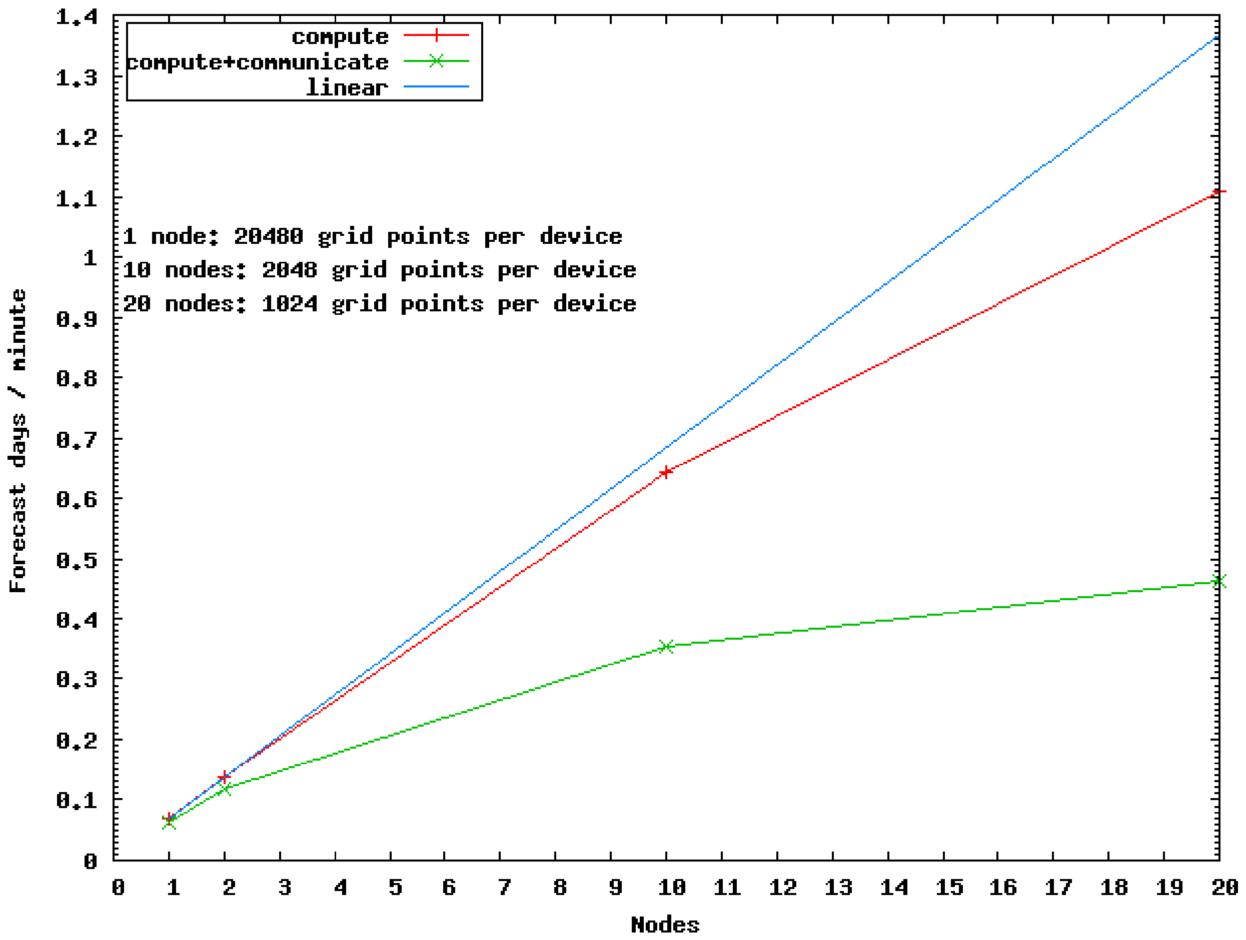
NIM thread scaling on SNB



Weak scaling on Phi



NIM strong scaling symmetric mode



Weak scaling on SNB