OPTIMIZING HPCG ON TIANHE2 AT THE FULL-SYSTEM SCALE

Chao Yang, Yutong Lu
CAS NUDT

Tianhe-2

- Compute node
  - Total number: 16000
  - Each with 2 12-core CPUs + 3 57-core MICs

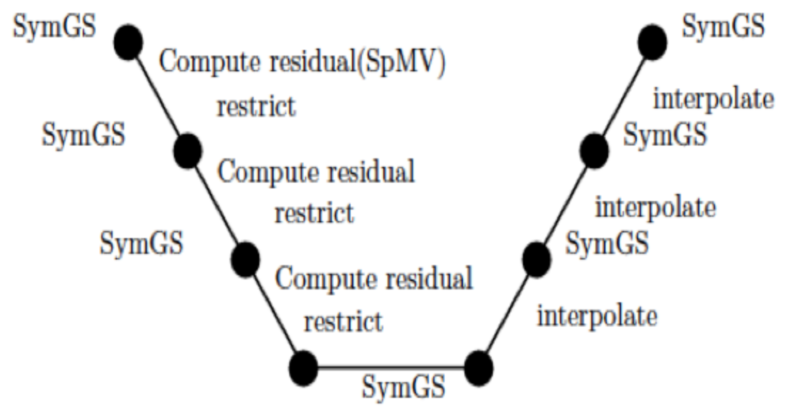
	CPU: Intel Xeon E5-2692	MIC: Intel Xeon Phi 31S1P
Frequency	2.2 GHz	1.1 GHz
Sockets/Cores/Threads	2 / 12 / 24	1 / 57 / 228
SIMD Width	4 double / 8 integer	8 double / 16 integer
Instruction Set	AVX	IMCI
Peak Gflops in D.P.	422.4	1003.2
L1 / L2 / L3 Cache	32 KB / 256 KB / 30 MB	32 KB / 512 KB / -
Memory Capacity	64 GB	8 GB
Stream Bandwidth	78 GB/s	180 GB/s

Algorithms in HPCG

- BLAS-1
  - WAXPBY (3)
    - w=a*x+b*y
  - DotProduct (3)
    - MPI_AllReduce comm
- Sparse BLAS-2
  - SpMV
    - MPI neighboring comm
- Preconditioner M^{-1}
  - SymGS
    - SpMV-like
    - MPI neighboring comm
  - Restriction
  - Prolongation

Algorithm 1 PCG for $Ax = b$

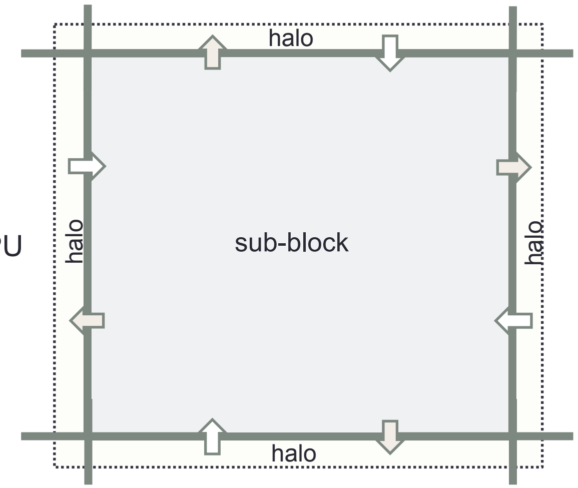
```
Input: $A$, $b$, $x_0$, $it_{max}$, $\varepsilon$
 1: $r_0 \leftarrow b - Ax_0$
 2: repeat $(i = 0, 1, \ldots)$
 3:     $z_i \leftarrow M^{-1} r_i$
 4:     $s_i \leftarrow (r_i, z_i)$
 5:     if $(i = 0)$   $p_i \leftarrow z_i$
 6:     else           $p_i \leftarrow z_i + (s_i / s_{i-1}) p_{i-1}$
 7:     $\alpha_i \leftarrow s_i / (p_i, Ap_i)$
 8:     $x_{i+1} \leftarrow x_i + \alpha_i p_i$
 9:     $r_{i+1} \leftarrow r_i - \alpha_i Ap_i$
10: until $(i + 1 = it_{max})$ or $(||r_{i+1}||_2 / ||r_0||_2 \leq \varepsilon)$
Output: $x_{i+1}$
```



V-cycle geometric multigrid algorithm

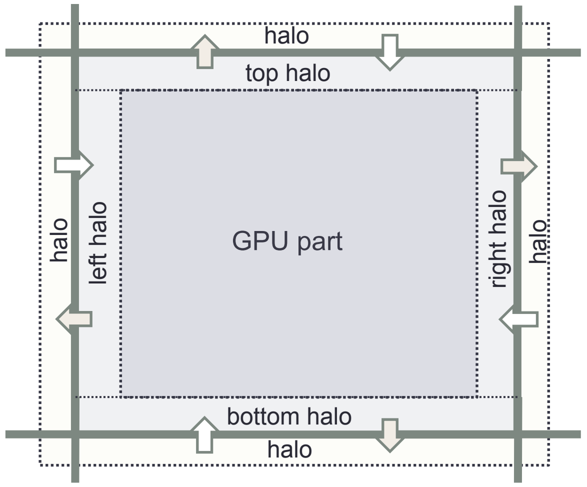
Heterogeneous Domain Decomposition

Pure CPU



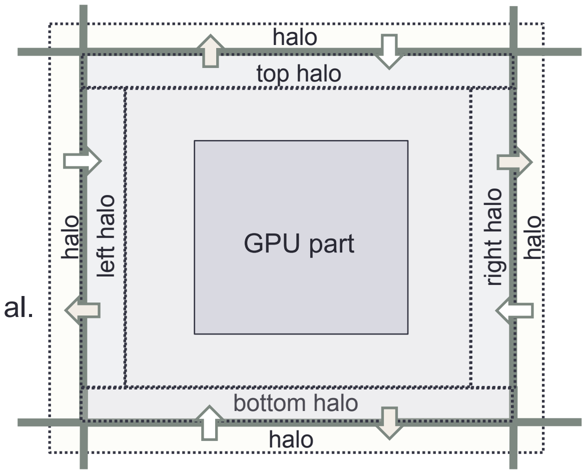
TSUBAME

SC'11
Shimokawabe
et al.



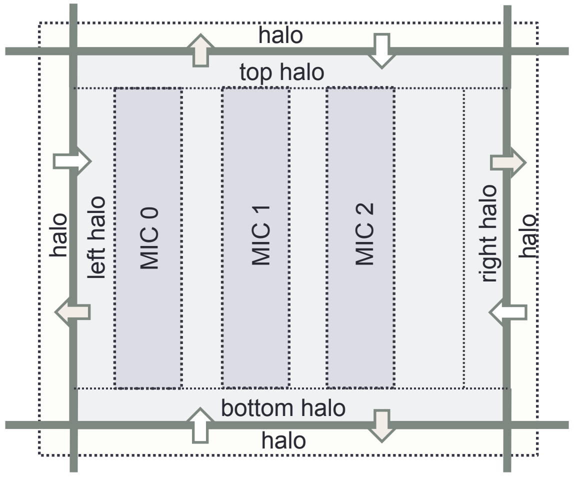
TH-1A

PPoPP'13
Yang, Xue, et al.



TH-2

IPDPS'14
IEEE TC'15
Xue, Yang, et al.



Overlapping comm. & comput.

- Each regular subdomain is divided into
  - Outer part → CPU
  - Inner part → Acclerator

ACC side:

① Compute inner part

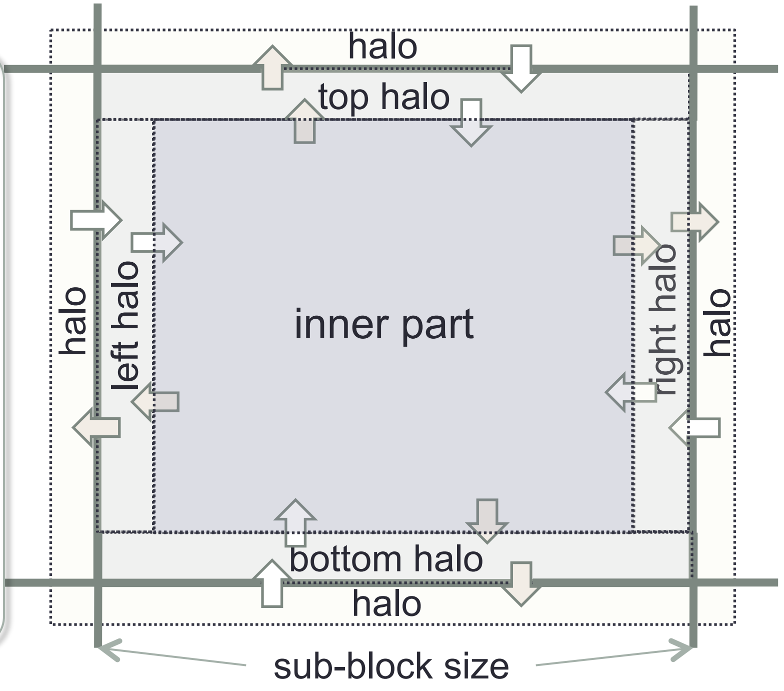
CPU side:

① Exchange halos

② Compute outer part

BARRIER:

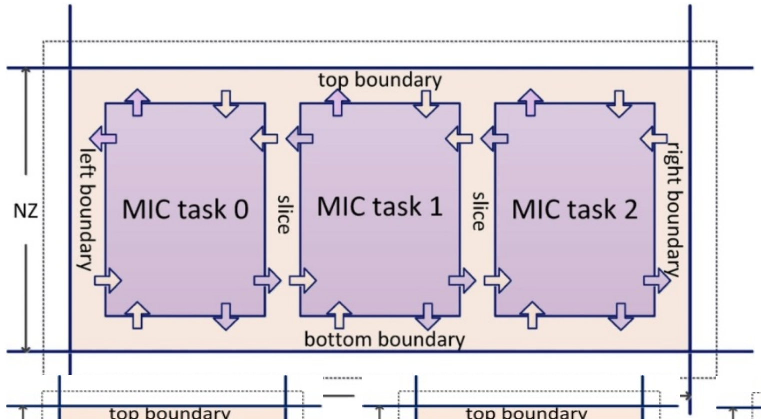
⑤ Exchange data between CPU/ACC



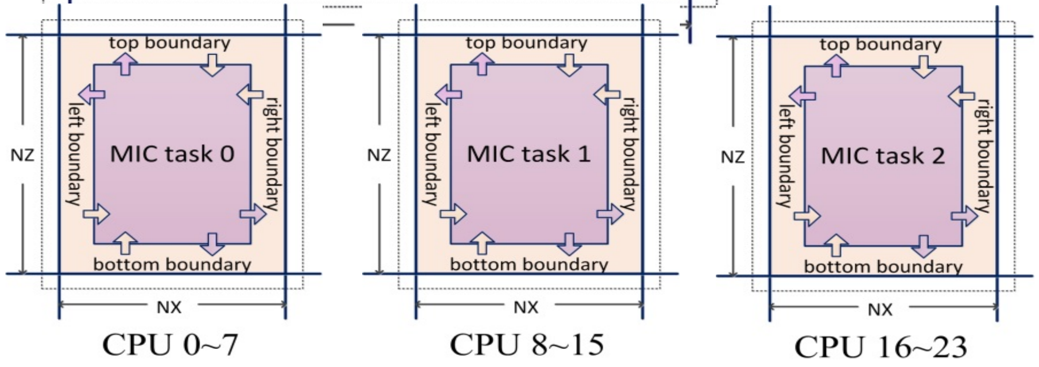
Heterogeneous DD for HPCG

- inner-outer domain decomposition
  - Pros
    - Isolated communication
      - No data transfer needed between inner parts
      - Inner parts are free of MPI communication
  - Cons
    - Outer part is irregular

✧ 1 MPI proc per node



✧ 3 MPI procs per node



Overhead of allocation in offload

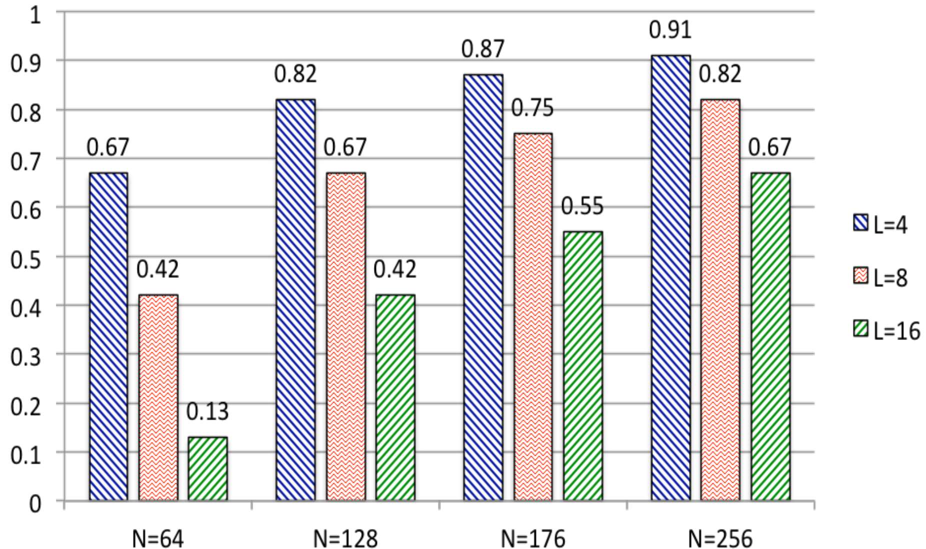
- Optimization phase
  - About 3GB data is allocated and transferred
- Unfortunate facts
  - MIC_TASK_NUM=1: cost=3s
  - MIC_TASK_NUM=3: cost=8s
  - MIC_TASK_NUM=3 (without #pragma omp parallel): cost=9s

```
#pragma omp parallel for num_threads(MIC_TASK_NUM)
   for (int i=0; i< MIC_TASK_NUM; i++) {
#pragma offload target(mic: i) \
      in(nnz_values:length(local_rows * nonzeros_in_row) ALLOC) \
      in(mtx_index:length(local_rows * nonzeros_in_row) ALLOC) \
      in(diagonal:length(local_rows) ALLOC) \
      in(sendindex:length(sendlength) ALLOC) \
      nocopy(sendbuffer:length(sendlength) ALLOC) \
      nocopy(recvbuffer:length(recvlength) ALLOC) \
      nocopy(counter:length(len) ALLOC) signal(...)
      { }
}
```

We have to use approach-2

Load balance

- Analysis on the finest level
  - Assume the subdomain size is $N_x N_y N_z$ and the thickness of outer is L
  - Effective ratio of inner part $R_{\text{effective}} = \dfrac{(N_x - 2L)(N_y - 2L)(N_z - 2L)}{N_x N_y N_z}$
  - Computing capacity $R_{\text{theoretical}} = \dfrac{\text{flops}_{\text{mic}}}{\text{flops}_{\text{cpu}} + \text{flops}_{\text{mic}}}$
  - Let $R_{\text{effective}} \gtrsim R_{\text{theoretical}} \approx 87.7\%$
  - A few typical values (right)
  - Conclusion
    - L=4 is the best choice

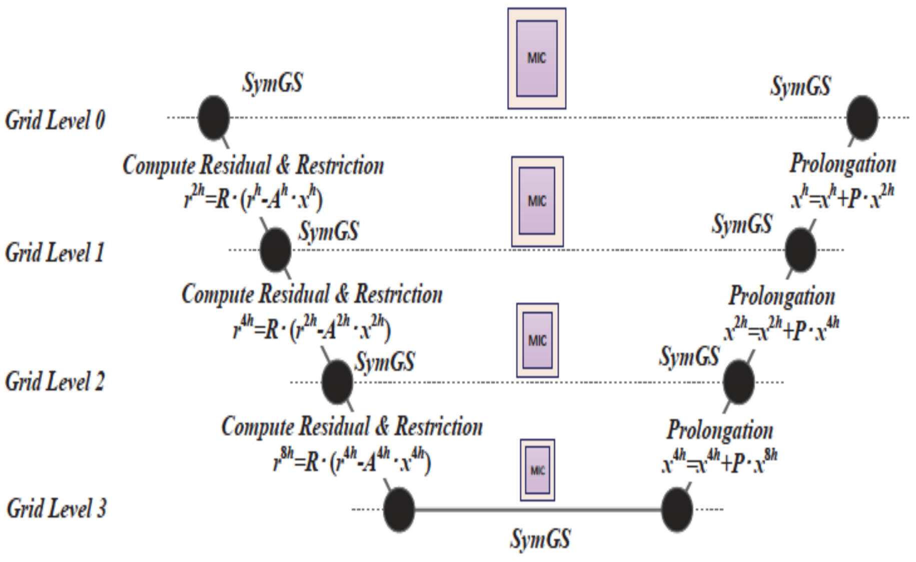


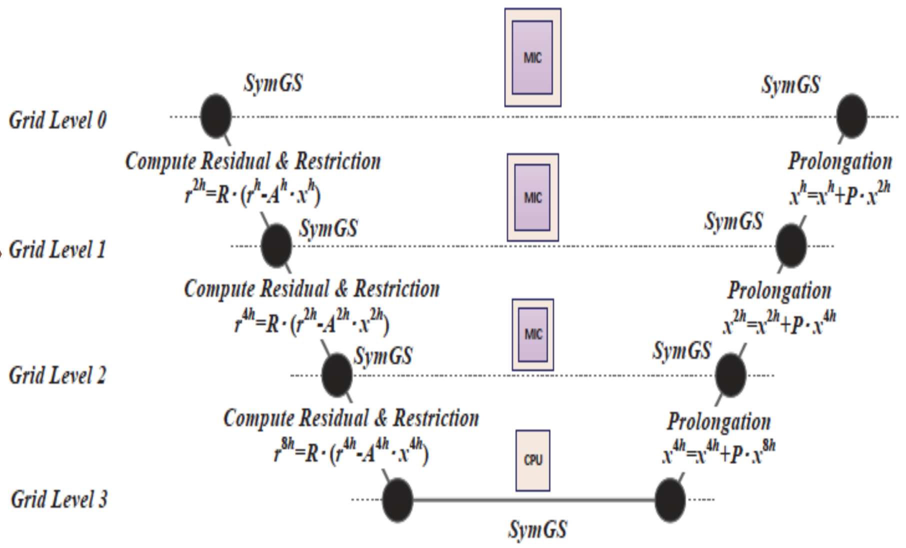
Load balance

- V-cycle geometric MG (4 levels)
  - Fine grid: (NX, NY, NZ) → coarse grid: (NX/2, NY/2, NZ/2)

Boundary thickness on finest level: L=8
- Minimize data transfer on all levels
- The outer part is too large on CPU

Boundary thickness on finest level: L=4
- Achieve a better load balance
- Transfer data to CPU on the coarsest level

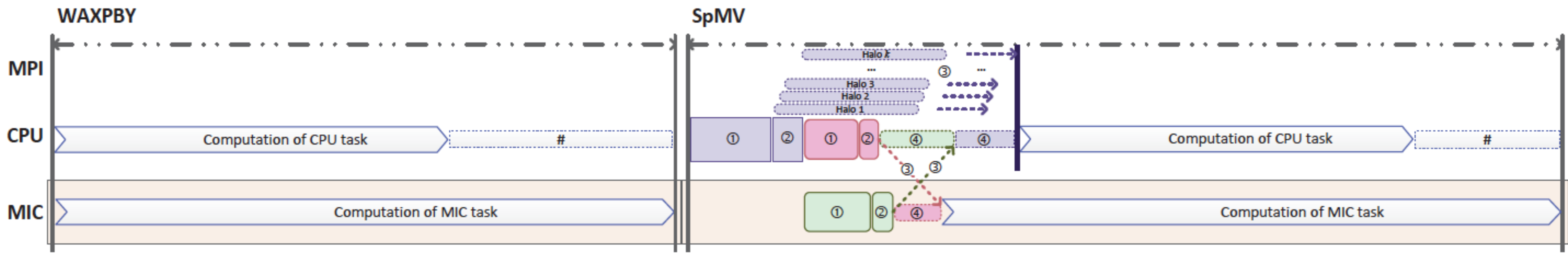

MIC
SymGS
Grid Level 0
Compute Residual & Restriction
$r^{2h}=R\cdot(r^h-A^h\cdot x^h)$
Prolongation
$x^h=x^h+P\cdot x^{2h}$
Grid Level 1
Compute Residual & Restriction
$r^{4h}=R\cdot(r^{2h}-A^{2h}\cdot x^{2h})$
Prolongation
$x^{2h}=x^{2h}+P\cdot x^{4h}$
Grid Level 2
Compute Residual & Restriction
$r^{8h}=R\cdot(r^{4h}-A^{4h}\cdot x^{4h})$
Prolongation
$x^{4h}=x^{4h}+P\cdot x^{8h}$
Grid Level 3
SymGS


MIC
SymGS
Grid Level 0
Compute Residual & Restriction
$r^{2h}=R\cdot(r^h-A^h\cdot x^h)$
Prolongation
$x^h=x^h+P\cdot x^{2h}$
Grid Level 1
Compute Residual & Restriction
$r^{4h}=R\cdot(r^{2h}-A^{2h}\cdot x^{2h})$
Prolongation
$x^{2h}=x^{2h}+P\cdot x^{4h}$
Grid Level 2
Compute Residual & Restriction
$r^{8h}=R\cdot(r^{4h}-A^{4h}\cdot x^{4h})$
Prolongation
$x^{4h}=x^{4h}+P\cdot x^{8h}$
CPU
Grid Level 3
SymGS

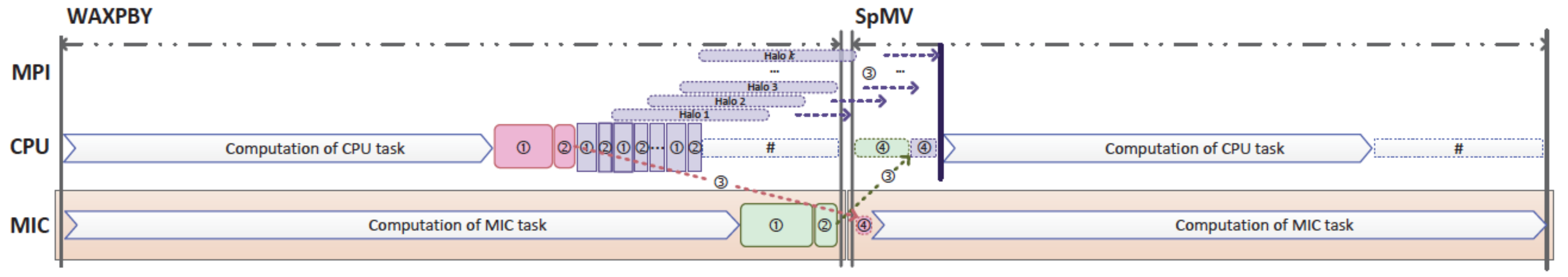
Asynchronous data movement

- Two types
  - MPI neighboring comm.
  - PCI-E data transfoer
- Four-step procedure

Preceding kernel	Current kernel	Calling position
WAXPBY	SpMV	CG
SymGS (Pre-smoother)	SpMV	MG (Level 0-1)
Prolongation	SymGS (Post-smoother)	MG (Level 0-2)



(a) The unfused schedule.



(b) The fused schedule.

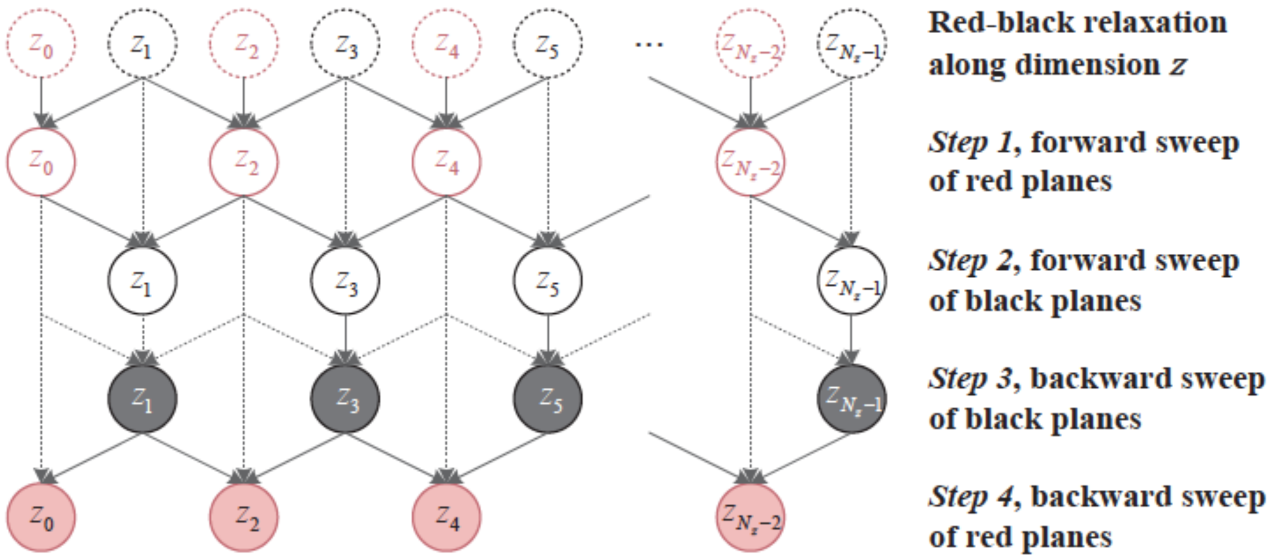
Optimizations: SpMV

- Sparse matrix format: CSR ->ELLPACK or SELLPACK

```
for (i=0; i< nrows/nb; i++) {
    register __m512d vsum0, vsum1, vx0, vx1, vv0, vv1;
    register __m512i vcol0, vcol1;
    vsum0 = _mm512_setzero_pd();
    vsum1 = _mm512_setzero_pd();
    for (int k = 0; k < nnzs_in_row; k++) {
        vcol0 = _mm512_load_epi32(&(cols[i×(nb×nnzs_in_row) + k×nb]));
        vcol1 = _mm512_permute4f128_epi32(vcol0, _MM_PERM_BADC);
        vx0 = _mm512_i32logather_pd(vcol0, x, sizeof(double));
        vx1 = _mm512_i32logather_pd(vcol1, x, sizeof(double));
        vv0 = _mm512_load_pd(&(nnz[i×(nb×nnzs_in_row) + k×nb]));
        vv1 = _mm512_load_pd(&(nnz[i×(nb×nnzs_in_row) + k×nb + 8]);
        vsum0 = _mm512_fmadd_pd(vx0, vv0, vsum0);
        vsum1 = _mm512_fmadd_pd(vx1, vv1, vsum1);
    }
    _mm512_storenrngo_pd(&(y[i×nb]), vsum0);
    _mm512_storenrngo_pd(&(y[i×nb + 8]), vsum1);
}
```

Optimizations: SymGS

- Symmetric Gauss-Seidel: SymGS(A,x,b)
  - Multi-coloring for fine-grained parallelism
    - Different strategies may result in different convergence behavior
  - Temporal locality
    - Exploit data reuse between forward/backward sweeps
  - Fused forward/backward sweep

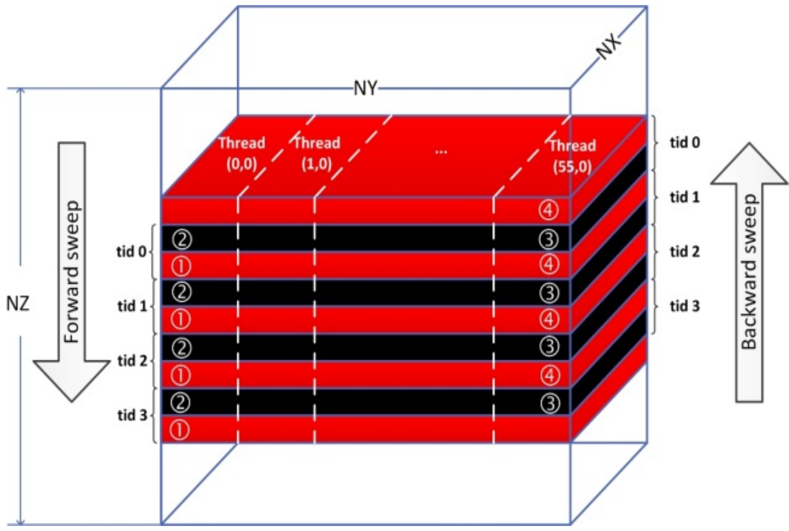

z_0
z_1
z_2
z_3
z_4
z_5
z_{N_z-2}
z_{N_z-1}
Red-black relaxation along dimension *z*
***Step 1*, forward sweep of red planes**
***Step 2*, forward sweep of black planes**
***Step 3*, backward sweep of black planes**
***Step 4*, backward sweep of red planes**

Optimizations: SymGS

- Symmetric Gauss-Seidel: SymGS(A,x,b)
  - MIC side: fused F/B GS + 2D thread grouping

```
forward sweep along $z_{i+2*tid}$
```

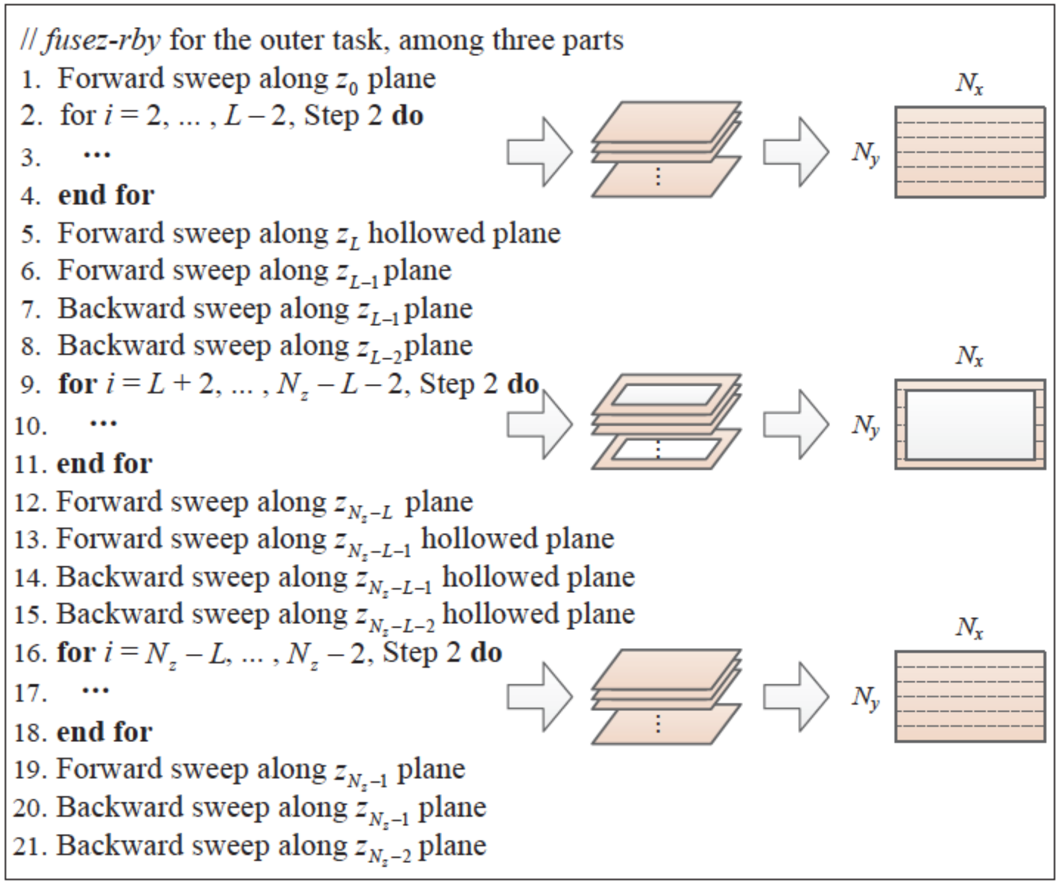
```
for $y_j$=beg_y,...,end_y-2, step 2 do
      forward sweep along ($z_{i+2*tid}$, $y_j$)
~~#pragma omp barrier~~ WaitNeighborCores

for $y_j$=beg_y+1,...,end_y-1,step 2 do
      forward sweep along ($z_{i+2*tid}$, $y_j$)
~~#pragma omp barrier~~ WaitNeighborCores
                        IntraBarrier
```



（b）fusez-rby-2level

Optimizations: SymGS

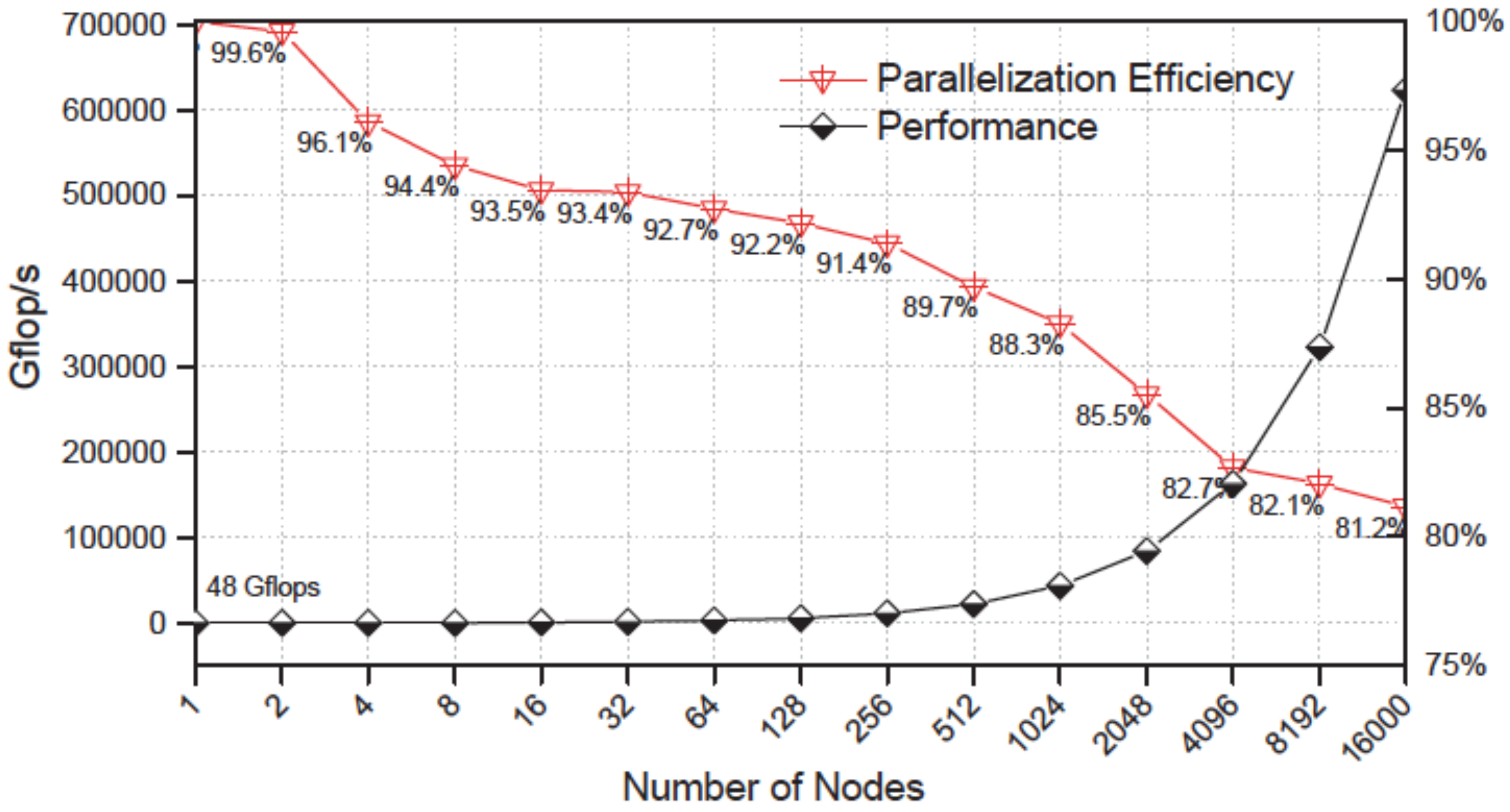
- Symmetric Gauss-Seidel: SymGS(A,x,b)
  - MIC side: fused F/B GS for irregular domain partition



```
// fusez-rby for the outer task, among three parts
1.  Forward sweep along $z_0$ plane
2.  for $i = 2, \dots, L-2$, Step 2 do
3.      ...
4.  end for
5.  Forward sweep along $z_L$ hollowed plane
6.  Forward sweep along $z_{L-1}$ plane
7.  Backward sweep along $z_{L-1}$ plane
8.  Backward sweep along $z_{L-2}$ plane
9.  for $i = L+2, \dots, N_z - L - 2$, Step 2 do
10.     ...
11. end for
12. Forward sweep along $z_{N_z-L}$ plane
13. Forward sweep along $z_{N_z-L-1}$ hollowed plane
14. Backward sweep along $z_{N_z-L-1}$ hollowed plane
15. Backward sweep along $z_{N_z-L-2}$ hollowed plane
16. for $i = N_z - L, \dots, N_z - 2$, Step 2 do
17.     ...
18. end for
19. Forward sweep along $z_{N_z-1}$ plane
20. Backward sweep along $z_{N_z-1}$ plane
21. Backward sweep along $z_{N_z-2}$ plane
```

Final full-system scale results

- Done on 11/18/2014



Some comments

- HPCG is highly bandwidth-limited and may be benefited
  - from data reuse (via register/cache etc), and/or
  - from hiding data movement (PCI-E transfer, halo exchange, etc).
- Enabling and scaling HPCG on heterogeneous systems is not easy because of possible
  - load imbalance among different computing units,
  - extra data movement between host and device, and
  - convergence penalty due to breaking of data dependency.
- The inner-outer domain decomposition method proves to be a good way to go due to
  - adjustable load balance,
  - reduced data movement, and
  - small convergence penalty.

THANK YOU!

MORE DETAILS CAN BE FOUND IN OUR IJHPCA PAPER

yangchao@iscas.ac.cn