Optimization of Communications towards Scalable Algorithms on Post Petascale Supercomputers

Kengo Nakajima
Information Technology Center, The University of Tokyo

ScalA15: Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems in conjunction with SC15
November 16, 2015, Austin, Texas
• ppOpen-HPC
• ppOpen-MATH
  – ppOpen-MATH/MG: Multigrid Solver
  – Target Problems, Computer Systems
  – Optimization of Serial Communication
  – Optimization of Parallel Comm. (I): CGA
  – Optimization of Parallel Comm. (II): hCGA
• Summary
Post-Peta CREST
Development of System Software Technologies for Post-Peta Scale High Performance Computing

- Objectives
  - Co-design of system software with applications and post-peta scale computer architectures
  - Development of deliverable software pieces
- Research Supervisor
  - Prof. Mitsuhisa Sato (RIKEN AICS)
- Run by JST (Japan Science and Technology Agency)
- Budget and Formation (2010 to 2018)
  - 55M-60M $ in total
  - Round 1: From 2010 for 5.5 year (5 Teams)
  - Round 2: From 2011 for 5.5 year (5 Teams)
  - Round 3: From 2012 for 5.5 year (4 Teams)

http://www.postpeta.jst.go.jp/en/
System Software

Takeshi Nanri, Kyushu University
Development of Scalable Communication Library with Technologies for Memory Saving and Runtime Optimization

Osamu Tatebe, U. of Tsukuba
System Software for Post Petascale Data Intensive Science

Taisuke Boku, U. of Tsukuba
Research and Development on Unified Environment of Accelerated Computing and Interconnection for Post-Petascale Era

Atsushi Hori, RIKEN AICS
Parallel System Software for Multi-core and Many-core

Masaaki Kondo, U. of Tokyo
Power Management Framework for Post-Petascale Supercomputers

c/o Y. Ishikawa (RIKEN)
Naoya Maruyama, Riken AICS
Highly Productive, High Performance Application Frameworks for Post Petascale Computing

Hiroyuki Takizawa, Tohoku University
An evolutionary approach to construction of a software development environment for massively-parallel heterogeneous systems

Shigeru Chiba, U. Tokyo
Software development for post petascale super computing --- Modularity for Super Computing

c/o Y. Ishikawa (RIKEN)
Applications & Numerical Libraries

Tetsuya Sakurai, University of Tsukuba
Development of an Eigen-Supercomputing Engine using a Post-Petascale Hierarchical Model

Kengo Nakajima, University of Tokyo
ppOpen-HPC: Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications with Automatic Tuning (AT)

Ryuji Shioya, Toyo University
Development of a Numerical Library based on Hierarchical Domain Decomposition for Post-Petascale Simulation

Katsuki Fujisawa, Kyushu University
Advanced Computing and Optimization Infrastructure for Extremely Large-Scale Graphs on Post Peta-Scale Supercomputers

Itsuki Noda, AIST
Framework for Administration of Social Simulations on Massively Parallel Computers

c/o Y. Ishikawa (RIKEN)
ppOpen-HPC: Overview

- Application framework with automatic tuning (AT)
  - “pp”: post-peta-scale
- Five-year project (FY.2011-2015) (since April 2011)
  - P.I.: Kengo Nakajima (ITC, The University of Tokyo)
  - Part of “Development of System Software Technologies for Post-Peta Scale High Performance Computing” funded by JST/CREST (Supervisor: Prof. Mitsuhisa Sato, Co-Director, RIKEN AICS)
- Team with 7 institutes, >50 people (5 PDs) from various fields: Co-Design
  - ITC/U.Tokyo, AORI/U.Tokyo, ERI/U.Tokyo, FS/U.Tokyo
  - Hokkaido U., Kyoto U., JAMSTEC
• **Group Leaders**
  – Masaki Sato (AORI/U.Tokyo)
  – Takashi Furumura (ERI/U.Tokyo)
  – Hiroshi Okuda (GSFS/U.Tokyo)
  – Takeshi Iwashita (Kyoto U., ITC/Hokkaido U.)
  – Hide Sakaguchi (IFREE/JAMSTEC)

• **Main Members**
  – Takahiro Katagiri (ITC/U.Tokyo)
  – Masaharu Matsumoto (ITC/U.Tokyo)
  – Hideyuki Jitsumoto (Tokyo Tech)
  – Satoshi Ohshima (ITC/U.Tokyo)
  – Hiroyasu Hasumi (AORI/U.Tokyo)
  – Takashi Arakawa (RIST)
  – Futoshi Mori (ERI/U.Tokyo)
  – Takeshi Kitayama (GSFS/U.Tokyo)
  – Akihiro Ida (ACCMS/Kyoto U.)
  – Miki Yamamoto (IFREE/JAMSTEC)
  – Daisuke Nishiura (IFREE/JAMSTEC)
ppOpen-HPC covers ...

- **FEM** (Finite Element Method)
- **FDM** (Finite Difference Method)
- **FVM** (Finite Volume Method)
- **BEM** (Boundary Element Method)
- **DEM** (Discrete Element Method)
Supercomputers in U.Tokyo

2 big systems, 6 yr. cycle

<table>
<thead>
<tr>
<th>FY</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>Hitachi SR11000/J2</td>
<td>18.8 TFLOPS, 16.4 TB</td>
<td>Fat nodes with large memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>Hitachi SR16000/M1</td>
<td>54.9 TFLOPS, 11.2 TB</td>
<td>Our last SMP, to be switched to MPP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>Hitachi HA8000 (T2K)</td>
<td>140 TFLOPS, 31.3 TB</td>
<td>(Flat) MPI, good comm. performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>Fujitsu PRIMEHPC FX10</td>
<td>1.13 PFLOPS, 150 TB</td>
<td>Turning point to Hybrid Parallel Prog. Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>Post T2K</td>
<td>25+ PFLOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Initial Plan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>京(=K)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Peta</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hitachi SR11000/J2
18.8 TFLOPS, 16.4 TB

Hitachi SR16000/M1
based on IBM Power-7
54.9 TFLOPS, 11.2 TB

Hitachi HA8000 (T2K)
140 TFLOPS, 31.3 TB

(Fat) MPI, good comm. performance

Fujitsu PRIMEHPC FX10
based on SPARC64 IXfx
1.13 PFLOPS, 150 TB

Turning point to Hybrid Parallel Prog. Model

Post T2K
25+ PFLOPS

Initial Plan

京(=K)

Peta
Target of ppOpen-HPC: Post T2K System

- Target system is Post T2K system
  - 25+ PFLOPS, FY.2016
    - JCAHPC (Joint Center for Advanced High Performance Computing): U. Tsukuba & U. Tokyo
    - http://jcahpc.jp/
  - Many-core based (e.g. Intel MIC/Xeon Phi)
    - MPI + OpenMP + X
  - ppOpen-HPC helps smooth transition of users (> 2,000) to new system

- K/FX10, Cray, Xeon clusters are also in scope
Schedule of Public Release
(with English Documents, MIT License)
http://ppopenhpc.cc.u-tokyo.ac.jp/

• Released at SC-XY (or can be downloaded)
• Multicore/manycore cluster version (Flat MPI, OpenMP/MPI Hybrid) with documents in English
• We are now focusing on MIC/Xeon Phi
• Collaborations are welcome

• History
  – SC12, Nov 2012 (Ver.0.1.0)
  – SC13, Nov 2013 (Ver.0.2.0)
  – SC14, Nov 2014 (Ver.0.3.0)
  – SC15, Nov 2015 (Ver.1.0.0)
New Features in Ver.1.0.0
http://ppopenhpc.cc.u-tokyo.ac.jp/

• HACApK library for H-matrix comp. in ppOpen-APPL/BEM (OpenMP/MPI Hybrid Version)
  – First Open Source Library by OpenMP/MPI Hybrid

• ppOpen-MATH/MP (Coupler for Multiphysics Simulations, Loose Coupling of FEM & FDM)

• Matrix Assembly and Linear Solvers for ppOpen-APPL/FVM
HACApK library

Library for simulations using the integral equation method

\[ g[u] = f, \text{ where } g[u](x) = \int_{\Omega} \kappa(x, y) u(y) \, dy \]

singular kernel: \( \kappa(x, y) \in \text{span}\{\{|x - y|^{-p}, p > 0\}\} \)

\( \kappa(x, y) \propto |x - y|^{-1} \)

For large-scaled simulations

- Approximation technique for matrices
  - H-matrices with ACA (Adaptive Cross Approximation)
    : \( O(N^2) \Rightarrow O(N\log N) \)

- Parallel computing
  - Hybrid MPI+OpenMP programming model

Download site: http://ppopenhpc.cc.u-tokyo.ac.jp
  - Open source  - MIT license

[A. Ida & T. Iwashita]
Overview of H-matrices with ACA

- Approximation technique for matrices from Integral operator.

\[ g[u](x) = \int_\Omega g(x, y) \, u(y) \, dy \]

singular kernel: \( g(x, y) \in \text{span}\{ |x - y|^{-p}, p > 0 \} \)

Discretization → H-matrices with ACA

- Low-rank matrix can be approximated by some pivot columns and rows.

ACA: Adaptive Cross Approximation

Full rank dense matrix

- Full-Rank
- Low-Rank

\[ m \approx n \]

\[ n \quad k \quad n \]

\[ k \]
Low-rank approximation using ACA

Applied to blocks detected as possible low-rank submatrices

**ACA**

\[ a \approx \tilde{a}_k = \sum_{\nu=1}^{k} \nu (w^\nu)^T \]

- Pivot column and pivot row are alternately selected vector by vector.
  \( \nu^1 \): arbitrary column (e.g. leftmost)
  \( w^1 \): \( j^1 \)th-row, \( j^1 := \arg\max_j |\nu^1_j| \)

- Approximation error estimation:
  \[
  \epsilon_k := \frac{||\nu^k|| \cdot ||w^k||}{\sqrt{\sum_{\nu=1}^{k} ||\nu|| \cdot ||w^\nu||}}
  \]

- Heuristic: \( \epsilon_k < \epsilon_{k-1} \)

Memory usage and approximation accuracy are controllable by the number of the selected vectors.
Example analysis using HACApK

Earthquake Cycle Simulation

Integral operator with $g(x, y) \propto |x - y|^{-3}$

- eq. motion
  $$\tau_i = \sum_{j=1}^{N} K_{ij} (u_j - V_{pt} t) - \frac{\zeta G}{2V_s} V_i$$

- friction law
  $$\tau_i = \mu \sigma_n^{eff} = \tau_* + A_i \ln(V_i / V_*) + B_i \ln(V*\theta_i / L_i)$$
  $$\frac{d\theta_i}{dt} = \exp(-V_i / V_c) - V_i \theta_i / L_i \ln(V_i \theta_i / L_i)$$

Subdivide fault surface
Example analysis using HACApK

Static electric field analysis

- Potential operator: \( V[u](x) := \int_{\partial \Omega} \frac{\nabla \cdot \mathbf{E}}{4\pi \| \mathbf{r} \|} \cdot \mathbf{n} \)

- Surface charge is calculated in half-infinite domain
Example analysis using HACApK

- **Static electric field analysis**
  - Potential operator:
    \[ V[u](x) := \int_{\Gamma} \frac{1}{4\pi \|x - y\|} u(y) \, dy, \quad x \in \Gamma \]

- Surface charge is calculated in half-infinite domain.

**Analysis condition**

**Numerical result**
Memory usage of HACApK and original dense matrices

- H-matrices with ACA reduce memory usage.

![Graph showing memory usage (Log-Log scale)](image)

- **Dense matrices**
- **HACApK** (Static electric field)
- **HACApK** (Earthquake cycle)

[A. Ida & T. Iwashita]
To apply H-matrices for huge-sized problems

Our efforts include:

■ Parallelization to exploit SMP cluster system
■ Improvement for large-sized problem

- Conventional H-matrices can fail to make efficient approximation when applied to large scale problem.

■ New algorithm of linear solver
  - BiCGSTAB and GCR are available.
  - Is any preconditioner needed?

[A. Ida & T. Iwashita]
Parallelization of H-matrices in HACApK

- **When constructing H-matrices**
  - Only step 3 (time-consuming part) is parallelized.
  - Any MPI communication is NOT needed.
  
  | step1 Make Cluster tree | Redundant computation on all MPI processors |
  | step2 Make H-matrix structure | parallel computing |
  | step3 Fill in sub-matrices(ACA) | |

- **When performing HMVM** (H-matrix-vector multiplication)
  - All MPI processes have the full multiplicand vector.
  - MPI communications are needed to realize it.

- **In both parallelization above**
  - Same assignment are used.
  - Arithmetic are conducted sub-matrix by sub-matrix.
  - Assignment to each process is a collection of sub-matrices.

[A. Ida & T. Iwashita]
Intention for assignment strategy

■ For MPI-processes

① Minimize the maximum \( R(M_k) \) as possible
⇒ Reducing transferred data size

② Minimize the load imbalance among MPI processes

■ For OpenMP-threads

・ Minimize the load imbalance among OpenMP-threads
Difference in assignment between strategies

Assigned submatrices to MPI-processes in HACApK

Assigned submatrices optimized for load balance

[A. Ida & T. Iwashita]
Performance test of HACApK

Parallel scalability is examined

- when constructing H-matrices
- when performing HMVM

Computer: Fujitsu FX10 at the university of Tokyo

Processor: SPARC64™ Ixfx (16cores/node)
Memory: 32GB
Network: 5 GB/s, Tofu.

The number of unknowns: \( N \)

- case 1: \( N = 1,000 \)
- case 2: \( N = 10,000 \)
- case 3: \( N = 100,000 \)
Parallel Scalability of $\mathcal{H}$AcApK(Flat-MPI)

- The larger the data size becomes, the better parallel scalability $\mathcal{H}$AcApK attains in both cases.
- Better parallel scalability is shown when constructing H-matrices.
- Parallel speed-up in a HMVM strongly depends on the data size.

![Graphs showing speed-up vs. number of processors.](image)

**Constructing H−matrices**

- 100,000 unknown
- 10,000 unknown
- 1,000 unknown

**H−matrix vector multiplication**

[A. Ida & T. Iwashita]
Effects of using Hybrid MPI+OpenMP in HMVM (FX10)

We examined speedup vs. the time of the Flat-MPI ver. on 1 node.

- Parallel scalability is improved in case of hybrid MPI+OpenMP by reducing MPI communication cost.
- Speed-up reaches a limit around 96-cores in case of Flat-MPI.

![Graph showing speed-up vs. number of cores for different configurations]

- 1,000,000 unknowns
- FX10

Parallel scalability when performing an H-matrix vector multiplication

[A. Ida & T. Iwashita]
Collaborations, Outreaching

- Collaborations
  - International Collaborations
    - Lawrence Berkeley National Lab.
    - National Taiwan University
    - ESSEX/SPPEXA/DFG, Germany
    - IPCC (Intel Parallel Computing Center)

- Outreaching, Applications
  - Large-Scale Simulations
    - Geologic CO$_2$ Storage
    - Astrophysics
    - Earthquake Simulations etc.
    - ppOpen-AT, ppOpen-MATH/VIS, ppOpen-MATH/MP, Linear Solvers
  - Intl. Workshops (2012, 13, 15)
  - Tutorials, Classes
• ppOpen-HPC
• ppOpen-MATH
  – ppOpen-MATH/MG: Multigrid Solver
  – Target Problems, Computer Systems
  – Optimization of Serial Communication
  – Optimization of Parallel Comm. (I): CGA
  – Optimization of Parallel Comm. (II): hCGA
• Summary
Sparse Linear Solvers in ppOpen-HPC

• (OpenMP+MPI) Hybrid
• Multicoloring/RCM/CM-RCM for OpenMP
  – Coloring procedures are NOT parallelized yet
• ppOpen-APPL/FEM, FVM, FDM
  – ILU/BILU(p,d,t)+CG/GPBiCG/GMRES, Depth of Overlapping
  – Hierarchical Interface Decomposition (HID) [Henon & Saad 2007], Extended HID [KN 2010]
• ppOpen-MATH/MG
  – Geometric Multigrid Solvers/Preconditioners
  – Comm./synch. avoiding/reducing based on hCGA
    • [KN 2014, Best Paper Award in IEEE/ICPADS 2014]
• ppOpen-APPL/BEM
  – H-Matrix Solver: HACApK
  – Only Open-Source H-Matrix Solver Library by OpenMP/MPI
User’s Program

ppOpen-APPL  FEM  FDM  FVM  BEM  DEM

ppOpen-MATH  MG  GRAPH  VIS  MP

ppOpen-AT  STATIC  DYNAMIC

ppOpen-SYS  COMM  FT

ppOpen-HPC

Optimized Application with
Optimized ppOpen-APPL, ppOpen-MATH
ppOpen-MATH

- A set of common numerical libraries
  - Multigrid solvers (ppOpen-MATH/MG)
  - Parallel graph libraries (ppOpen-MATH/GRAPH)
    - Multithreaded RCM for reordering (under development)
  - Parallel visualization (ppOpen-MATH/VIS)
  - Library for coupled multi-physics simulations (loose-coupling) (ppOpen-MATH/MP)
    - Originally developed as a coupler for NICAM (atmosphere, unstructured), and COCO (ocean, structured) in global climate simulations using K computer
      - Both codes are major codes on the K computer.
        » Prof. Masaki Satoh (AORI/U.Tokyo): NICAM
        » Prof. Hiroyasu Hasumi (AORI/U.Tokyo): COCO
    - Developed coupler is extended to more general use.
      - Coupled seismic simulations
pGW3D-FVM with ppOpen-MATH/MG

- 3D Groundwater Flow via Heterogeneous Porous Media
  - Poisson’s equation $\nabla \cdot (\lambda(x, y, z) \nabla \phi) = q$
  - Randomly distributed water conductivity
  - Finite-Volume Method on Cubic Voxel Mesh
    - $\lambda = 10^{-5} \sim 10^5$, Average: 1.00
    - MGCG solver with IC(0) smoother

- Multigrid
  - Scalable, one of the choices for post-peta/exascale HPC
  - HPCG
Linear Solvers

• Preconditioned CG Method
  – (Geometric) Multigrid Preconditioning (MGCG)
  – IC(0) for Smoothing Operator (Smotherer): good for ill-conditioned problems

• Parallel Geometric Multigrid Method
  – 8 fine meshes (children) form 1 coarse mesh (parent) in isotropic manner (octree)
  – V-cycle
  – Domain-Decomposition-based: Localized Block-Jacobi, Overlapped Additive Schwartz Domain Decomposition (ASDD)
  – Operations using a single core at the coarsest level (redundant)
Computations on Fujitsu FX10

• Fujitsu PRIMEHPC FX10 at U.Tokyo (Oakleaf-FX)
  – Commercial version of K
  – 16 cores/node, flat/uniform access to memory
  – 4,800 nodes 1.043 PF (74th, TOP 500, 2015 Nov.)

• Up to 4,096 nodes (65,536 cores) (Large-Scale HPC Challenge)
  – Max 17,179,869,184 unknowns
  – Flat MPI, HB 4x4, HB 8x2, HB 16x1

• Weak Scaling
• Strong Scaling
  – $128^3 \times 8 = 16,777,216$ unknowns, from 8 to 4,096 nodes
• Network Topology is not specified
  – 1D
HB M x N

Number of OpenMP threads per a single MPI process

Number of MPI process per a single node

Elements in “same color” are independent: to be parallelized by OpenMP on each MPI process.

MC (Color#=4)
Multicoloring

RCM
Reverse Cuthill-McKee

CM-RCM (Color#=4)
Cyclic MC + RCM
• **MC**: Good parallel efficiency with smaller # of colors, bad convergence. Better convergence with many colors, synch. overhead
• **RCM**: Good convergence, poor parallel efficiency, synch. overhead
• **CM-RCM**: Reasonable convergence & efficiency
Communications in MGCG are expensive!

- **Serial Communications**
  - Data Transfer through Memory Hierarchy
    - Sparse Matrix Operations in Parallel MG

- **Parallel Communications**
  - Message Passing through Network
pGW3D-FVM with ppOpen-MATH/MG

• Storage format of coefficient matrices (Serial Communication)
  – CRS (Compressed Row Storage)
  – ELL (Ellpack-Itpack)

• Communication/Synchronization Reducing MG (Parallel Communication)
  – Coarse Grid Aggregation (CGA)
  – Hierarchical CGA: Comm. Reducing CGA
• ppOpen-HPC
• ppOpen-MATH
  – ppOpen-MATH/MG: Multigrid Solver
  – Target Problems, Computer Systems
  – **Optimization of Serial Communication**
  – Optimization of Parallel Comm. (I): CGA
  – Optimization of Parallel Comm. (II): hCGA
• Summary
ELL: Fixed Loop-length, Nice for Pre-fetching (if ROW major)

(a) CRS

(b) ELL

Additional Memory & Computations
ELL with Row-wise Sweeping
CRS with fixed length
Backward Substitution

!$omp parallel
  do icol= 1, NCOLORtot
    !$omp do
      do ip = 1, PEsmptot
        do i= Index(ip-1,icol)+1, Index(ip,icol)
          do k= 1, 6
            Z(i)= Z(i) - AMU(k,i)*Z(IAMU(k,i))
          enddo
          Z(i)= Z(i) / DD(i)
        enddo
      enddo
    !$omp end do
  enddo
!$omp end parallel
Special Treatment for “Boundary” Meshes connected to “Halo”

- Distribution of Lower/Upper Non-Zero Off-Diagonal Components
- If we adopt RCM (or CM) reordering ...
- Pure Internal Meshes
  - L: ~3, U: ~3
- Boundary Meshes
  - L: ~3, U: ~6
Original ELL: Backward Substitution
Number of Non-Zero Off-Diag. Components for Upper Tri. Part
Cache is not well-utilized: IAU\text{new}(6,N), Aunew(6,N)
do icol = NHYP(lev), 1, -1
    if (mod(icol,2).eq.1) then
        !$omp parallel do private (ip,icel,j,SW)
            do ip = 1, PEsmpTOT
                do icel = SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
                    SW = 0.0d0
                    do j = 1, 6
                        SW = SW + AUnew(j,icel)*Rmg(IAUnew(j,icel))
                    enddo
                    Rmg(icel) = Rmg(icel) - SW*DDmg(icel)
                enddo
            enddo
        endif
    else
        !$omp parallel do private (ip,icel,j,SW)
            do ip = 1, PEsmpTOT
                do icel = SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
                    SW = 0.0d0
                    do j = 1, 3
                        SW = SW + AUnew(j,icel)*Rmg(IAUnew(j,icel))
                    enddo
                    Rmg(icel) = Rmg(icel) - SW*DDmg(icel)
                enddo
            enddo
        endif
    enddo
Improved ELL: Backward Substitution

Separate Arrays Introduced

Cache is well-utilized: AUnew3/AUnew6

Sliced ELL [Monakov et al. 2010] (for SpMV/GPU)
do icol = NHYP(lev), 1, -1
  if (mod(icol,2).eq.1) then
    !$omp parallel do private (ip,icel,j,SW)
    do ip = 1, PEsmpTOT
      do icel = SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
        SW = 0.0d0
        do j = 1, 6
          SW = SW + AUnew6(j,icel)*Rmg(IAUnew6(j,icel))
        enddo
        Rmg(icel) = Rmg(icel) - SW*DDmg(icel)
      enddo
    enddo
  else
    !$omp parallel do private (ip,icel,j,SW)
    do ip = 1, PEsmpTOT
      do icel = SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
        SW = 0.0d0
        do j = 1, 3
          SW = SW + AUnew3(j,icel)*Rmg(IAUnew3(j,icel))
        enddo
        Rmg(icel) = Rmg(icel) - SW*DDmg(icel)
      enddo
    enddo
  endif
enddo
There are a lot of “X”-ELL’s

• Mainly focusing on SpMV computations
• SELL-C-\(\sigma\)
• Recently, “X”-ELL’s are applied to forward/backward substitutions with data dependency
  – Most of HPCG implementations: SC14 BoF
  – They are focusing on Gauss-Seidel: much easier
• ILU
  – Upper/lower components must be treated separately
  – More difficult, complicated
  – (In this case L/U components are separately stored)
Analyses by Detailed Profiler of Fujitsu FX10, single node, Flat MPI, RCM (Multigrid Part), $64^3$ cells/core, 1-node

<table>
<thead>
<tr>
<th></th>
<th>Instruction</th>
<th>L1D miss</th>
<th>L2 miss</th>
<th>SIMD Op. Ratio</th>
<th>GFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRS</td>
<td>$1.53 \times 10^9$</td>
<td>$2.32 \times 10^7$</td>
<td>$1.67 \times 10^7$</td>
<td>30.14%</td>
<td>6.05</td>
</tr>
<tr>
<td>Original ELL</td>
<td>$4.91 \times 10^8$</td>
<td>$1.67 \times 10^7$</td>
<td>$1.27 \times 10^7$</td>
<td>93.88%</td>
<td>6.99</td>
</tr>
<tr>
<td>Improved ELL</td>
<td>$4.91 \times 10^8$</td>
<td>$1.67 \times 10^7$</td>
<td>$9.14 \times 10^6$</td>
<td>93.88%</td>
<td>8.56</td>
</tr>
</tbody>
</table>
• ppOpen-HPC
• ppOpen-MATH
  – ppOpen-MATH/MG: Multigrid Solver
  – Target Problems, Computer Systems
  – Optimization of Serial Communication
  – Optimization of Parallel Comm. (I): CGA
  – Optimization of Parallel Comm. (II): hCGA
• Summary
Parallel Multigrid: Original Approach

Coarse grid solver at a single core [KN 2010]

Communication Overhead at Coarser Levels

Coarse grid solver on a single core (further multigrid)
Coarse Grid Aggregation (CGA)

Coarse Grid Solver is multithreaded [KN 2012]

- Communication overhead could be reduced
- Coarse grid solver is more expensive than original approach.
- If process number is larger, this effect might be significant

Coarse grid solver on a single MPI process (multithreaded, further multigrid)
Weak Scaling: ~4,096 nodes
up to 17,179,869,184 meshes (64^3 meshes/core)

DOWN is GOOD

<table>
<thead>
<tr>
<th>CORE#</th>
<th>Matrix</th>
<th>Coarse Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>CRS</td>
<td>Single Core</td>
</tr>
<tr>
<td>C1</td>
<td>ELL (org)</td>
<td>Single Core</td>
</tr>
<tr>
<td>C2</td>
<td>ELL (org)</td>
<td>CGA</td>
</tr>
<tr>
<td>C3</td>
<td>ELL (sliced)</td>
<td>CGA</td>
</tr>
</tbody>
</table>
Weak Scaling: ~4,096 nodes
up to 17,179,869,184 meshes (64³ meshes/core)

DOWN is GOOD

<table>
<thead>
<tr>
<th></th>
<th>Matrix</th>
<th>Coarse Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>CRS</td>
<td>Single Core</td>
</tr>
<tr>
<td>C1</td>
<td>ELL (org)</td>
<td>Single Core</td>
</tr>
<tr>
<td>C2</td>
<td>ELL (org)</td>
<td>CGA</td>
</tr>
<tr>
<td>C3</td>
<td>ELL (sliced)</td>
<td>CGA</td>
</tr>
</tbody>
</table>

Matrix Coarse Grid

C0 CRS Single Core
C1 ELL (org) Single Core
C2 ELL (org) CGA
C3 ELL (sliced) CGA
Weak Scaling: C3
Results at 4,096 nodes (1.72x10^{10} DOF)
Weak Scaling: C2 (with CGA)

Time for Coarse Grid Solver

Efficiency of coarse grid solver for HB 16x1 is x256 of that of flat MPI (1/16 problem size, x16 resource for coarse grid solver)
Summary so far ...

- "Coarse Grid Aggregation (CGA)" is effective for stabilization of convergence at $O(10^4)$ cores for MGCG
  - Smaller number of parallel domains
  - HB 8x2 is the best at 4,096 nodes
  - Flat MPI, HB 4x4
    - Coarse grid solvers are more expensive, because their number of MPI processes are more than those of HB 8x2 and HB 16x1.

- ELL format is effective!
  - C0 (CRS) $\rightarrow$ C1 (ELL-org.): +20-30%
  - C2 (ELL-org) $\rightarrow$ C3(ELL-new): +20-30%
  - C0 $\rightarrow$ C3: +80-90%

- Coarse Grid Solver
  - Very expensive for cases with more than $O(10^5)$ cores
  - Memory of a single node is not enough
  - Multiple nodes should be utilized for coarse grid solver
• ppOpen-HPC
• ppOpen-MATH
  – ppOpen-MATH/MG: Multigrid Solver
  – Target Problems, Computer Systems
  – Optimization of Serial Communication
  – Optimization of Parallel Comm. (I): CGA
  – Optimization of Parallel Comm. (II): hCGA
• Summary
Hierarchical CGA: Comm. Reducing MG

Reduced number of MPI processes [KN 2013]

Fine

Level=1

Level=2

Level=m-3

Level=m-3

Level=m-2

Coarse

Coarse grid solver on a single MPI process (multi-threaded, further multigrid)
hCGA: Related Work

• Not a new idea, but very few implementations.
  – Not effective for peta-scale systems (Dr. U.M.Yang (LLNL), developer of Hypre)

• Existing Works: Repartitioning at Coarse Levels
  – Flat MPI, Repartitioning if DOF < O(10^3) on each process
**hCGA in the present work**

- Accelerate the coarser grid solver
  - using multiple processes instead of a single process in CGA
  - Only 64 cells on each process of lev=6 in the figure

- Straightforward Approach
  - MPI_Comm_split, MPI_Gather, MPI_Bcast etc.
Weak Scaling:
~4,096 nodes
up to 17,179,869,184 meshes
(64^3 meshes/core)
DOWN is GOOD

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Coarse Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>CRS</td>
</tr>
<tr>
<td>C1</td>
<td>ELL (org)</td>
</tr>
<tr>
<td>C2</td>
<td>ELL (org)</td>
</tr>
<tr>
<td>C3</td>
<td>ELL (sliced)</td>
</tr>
<tr>
<td>C4</td>
<td>ELL (sliced)</td>
</tr>
</tbody>
</table>

**Table:**
- Flat MPI
- HB 4x4
- HB 8x2
- HB 16x1

**Graph:**
- C3, 4,096 nodes
- C4, 4,096 nodes

**Legend:**
- Flat MPI:C3
- Flat MPI:C4
- HB 4x4:C4
- HB 8x2:C3
- HB 16x1:C3

**Graph:**
- Core Number
- Seconds

**Note:**
- x1.61
Optimum Parameters at 4,096 nodes
Weak Scaling

- Optimum level for switching to reduced number of MPI processes for CGA ($lev_{CGAopt}$) and $h$CGA ($lev_{hCGAopt}$)
- $N@lev_{CGAopt}, N@lev_{hCGAopt}$
  - Number of unknowns per each MPI process at the switching level (much smaller than $O(10^3)$ used in related works)
- Optimum # of MPI processes after repartitioning ($PE_{rep}$)

<table>
<thead>
<tr>
<th></th>
<th>lev$_{CGAopt}$</th>
<th>N@lev$_{CGAopt}$</th>
<th>lev$_{hCGAopt}$</th>
<th>N@lev$_{hCGAopt}$</th>
<th>PE$_{rep}$</th>
<th>Iter’s</th>
<th>sec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat MPI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>64</td>
<td>13.2</td>
</tr>
<tr>
<td>C4</td>
<td>6</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>61</td>
<td>8.22</td>
</tr>
<tr>
<td>HB 4x4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>8</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>59</td>
<td>8.08</td>
</tr>
<tr>
<td>C4</td>
<td>6</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td>56</td>
<td>7.97</td>
</tr>
</tbody>
</table>
CGA \( \text{lev}_{\text{CGAopt}}, N@\text{lev}_{\text{CGAopt}} \)

hCGA \( \text{lev}_{\text{hCGAopt}}, N@\text{lev}_{\text{hCGAopt}} \)
**Strong Scaling at 4,096 nodes**

268,435,456 meshes, $16^3$ meshes/core at 4,096 nodes

**UP is GOOD**

---

**Flat MPI/ELL (C3), 8 nodes (128 cores) : 100%**

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Coarse Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>CRS</td>
</tr>
<tr>
<td></td>
<td>Single Core</td>
</tr>
<tr>
<td>C1</td>
<td>ELL (org)</td>
</tr>
<tr>
<td></td>
<td>Single Core</td>
</tr>
<tr>
<td>C2</td>
<td>ELL (org)</td>
</tr>
<tr>
<td></td>
<td>CGA</td>
</tr>
<tr>
<td>C3</td>
<td>ELL (sliced)</td>
</tr>
<tr>
<td></td>
<td>CGA</td>
</tr>
<tr>
<td>C4</td>
<td>ELL (sliced)</td>
</tr>
<tr>
<td></td>
<td>hCGA</td>
</tr>
</tbody>
</table>

---

![Graph showing parallel performance](image-url)
• ppOpen-HPC
• ppOpen-MATH
  – ppOpen-MATH/MG: Multigrid Solver
  – Target Problems, Computer Systems
  – Optimization of Serial Communication
  – Optimization of Parallel Comm. (I): CGA
  – Optimization of Parallel Comm. (II): $h$CGA

• Summary
Summary

• \( hCGA \) is effective, but not so significant (except flat MPI)
  – flat MPI: \( x1.61 \) for weak scaling, \( x6.27 \) for strong scaling at 4,096 nodes of Fujitsu FX10
  – \( hCGA \) will be effective for HB 16x1 with more than \( 2.50 \times 10^5 \) nodes (\( = 4.00 \times 10^6 \) cores) of FX10 (\( = 60 \) PFLOPS)
    • Comp. time of coarse grid solver is significant for Flat MPI with \( >10^3 \) nodes
  – Communication overhead has been (slightly) reduced by \( hCGA \)
Future Works, Open Problems

• Improvement of $h$CGA
  – Overhead by MPI_Allreduce etc. -> P2P comm.: Put-Get

• Algorithms
  – CA-Multigrid (for coarser levels), CA-SPAI, Pipelined Method (Tianhe-2)

• Strategy for Automatic Selection
  – switching level, number of processes for $h$CGA, optimum color #
  – effects on convergence

• More Flexible ELL for Unstructured Grids
  – SELL-C-\(\sigma\)

• Xeon Phi Clusters
Number of Colors and Comp. Time

- ICCG Solvers
- FX10
- Ivy-Bridge (IvyB)
- KNC (MIC)

- “Optimum” number for each architecture is different
Overhead by Collective Comm.

- Overhead by global collective comm. (e.g. MPI_Allreduce)
- Change original Krylov solver so that comm. overhead by global coll. comm. are hidden by overlapping with other computations (Gropp’s asynch. CG, s-step, pipelined ...)
- “MPI_Allreduce” in MPI-3: MPI-3 on FX10, December 2015

Algorithm 7. Gropp’s asynchronous CG

1: \( r_0 := b - Ax_0; u_0 := M^{-1}r_0; p_0 := u_0; s_0 := Ap_0; \gamma_0 := (r_0, u_0) \)
2: for \( i = 0, \ldots \)
3: \( \delta := (p_i, s_i) \)
4: \( q_i := M^{-1}s_i \)
5: \( \alpha_i := \gamma_i / \delta \)
6: \( x_{i+1} := x_i + \alpha_ip_i \)
7: \( r_{i+1} := r_i - \alpha_is_i \)
8: \( u_{i+1} := u_i - \alpha_iq_i \)
9: \( q_{i+1} := (r_{i+1}, u_{i+1}) \)
10: \( w_{i+1} := Au_{i+1} \)
11: \( p_{i+1} := q_{i+1} / q_i \)
12: \( p_{i+1} := u_{i+1} + \beta_{i+1}p_i \)
13: \( s_{i+1} := w_{i+1} + \beta_{i+1}s_i \)
14: end for
SELL-C-\( \sigma \) for PCG in FEM Intel Xeon Phi (KNC)

```
Constructing SELL-C-\( \sigma \)
1. Pick chunk size \( C \) (guided by SIMD/T widths)
2. Pick sorting scope \( \sigma \)
3. Sort rows by length within each sorting scope
4. Pad chunks with zeros to make them rectangular
5. Store matrix data in “chunk column major order”

“Chunk occupancy”: fraction of “useful” matrix entries

\[
\beta = \frac{N_{use}}{\sum_{i=0}^{N_{col}} C \cdot l_i}
\]

```

![Graphs and equations related to SELL-C-\( \sigma \) performance metrics.](image)
Next Stage of ppOpen-HPC

• FY.2016-FY.2018
  – JST/CREST & DFG/SPPEXA (Germany) Collaboration
  – ESSEX: Equipping Sparse Solvers for Exascale
    • http://blogs.fau.de/essex/
    • Leading PI: Prof. Gerhard Wellein (U. Erlangen)
  – ESSEX II: ESSEX, Sakurai-T, Nakajima-T
    • Iterative Solver for Quantum Chemistry: pK-Open-SOL
      – Multgrid/Low-Rank Approximation
      – DLR (German Aerospace Research Center)
    • Performance Model for Stencil Computation: pK-Open-AT
      – U. Erlangen
      – kerncraft: Loop Kernel Analysis and Performance Modeling Toolkit
        » https://github.com/cod3monk/kerncraft
Please visit the booth of Oakleaf/Kashiwa Alliance, the University of Tokyo #2203