Scalable implementation of the parallel multigrid method on massively parallel computers

Kab Seok Kang 
kskang@ipp.mpg.de

High Level Support Team (HLST)
Department of Computational Plasma Physics
Max-Planck-Institut für Plasmaphysik,
Boltzmannstraße 2, D-85748 Garching, Germany

July 15, 2015
GIST HPC Summer school, Gwangju, Korea

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.
1. Computational Science and Engineering (CSE)
   - PDE based simulations

2. Multigrid Method
   - Basic idea
   - Parallelization issues and gathering data
   - OpenMP/MPI hybridization

3. Model problem
   - Model problem
   - Discretization
   - Parallelization

4. Numerical results
   - Helios Computer
   - Pure MPI
   - OpenMP/MPI hybridization

5. Conclusions
– A third mode of science, complementing and adding to experimentation/observation and theory
– Integral part of science and engineering research
– Multidisciplinary: the interface between physics, mathematics and computer science
High-Performance Computer (HPC) is an essential tool.

Figure 4. The process of scientific simulation, showing the validation and verification loop (left) and algorithm and performance tuning loop (right).
The five rules

– Rolls in CSE: Where to expert?
- Applicationist: Phenomenon, Physics, Mathematics (continuous), Analysis, Verification
- Mathematician: Mathematics (continuous), Numeric (discret), Algorithm, Solution, Analysis
- Computer scientist: Algorithm, Implementation and optimization, Port, Solution

– Know the physics: scientists and engineers know their problems well and how to build the mathematical model representing their physical problem
– Control the software
– Understand the numerics: less well understood by the scientific and engineering community.

– Considering two aspects of algorithm development:
- Do the numerics accurately capture the physical phenomena?
- Is the algorithm appropriate for parallel computing?

– Achieve expected behavior: testing, validation, and verification
– Question unexpected behavior → Collaborate applicationist, mathematician, and computer scientist
The five rules

– Rolls in CSE: Where to expert?
- Applicationist: Phenomenon, Physics, Mathematics (continuous), Analysis, Verification
- Mathematicain: Mathematics (continuous), Numeric (discret), Algorithm, Solution, Analysis

– Know the physics: scientists and engineers know their problems well and how to build the mathematical model representing their physical problem
– Control the software
– Understand the numerics: less well understood by the scientific and engineering community.

Considering two aspects of algorithm development:
- Do the numerics accurately capture the physical phenomena?
- Is the algorithm appropriate for parallel computing?

– Achieve expected behavior: testing, validation, and verification
– Question unexpected behavior → Collaborate applicationist, mathematician, and computer scientist
The five rules

– Rolls in CSE: Where to expert?
  - Applicationist: Phenomenon, Physics, Mathematics (continuous), Analysis, Verification
  - Mathematician: Mathematics (continuous), Numeric (discret), Algorithm, Solution, Analysis
  - Computer scientist: Algorithm, Implementation and optimization, Port, Solution

– Know the physics: scientists and engineers know their problems well and how to build the mathematical model representing their physical problem

– Control the software

– Understand the numerics: less well understood by the scientific and engineering community.

– Considering two aspects of algorithm development:
  - Do the numerics accurately capture the physical phenomena?
  - Is the algorithm appropriate for parallel computing?

– Achieve expected behavior: testing, validation, and verification

– Question unexpected behavior → Collaborate applicationist, mathematician, and computer scientist
The five rules

- Rolls in CSE: Where to expert?
  - Applicationist: Phenomenon, Physics, Mathematics (continuous), Analysis, Verification
  - Mathematician: Mathematics (continuous), Numeric (discret), Algorithm, Solution, Analysis
  - Computer scientist: Algorithm, Implementation and optimization, Port, Solution

- Know the physics: scientists and engineers know their problems well and how to build the mathematical model representing their physical problem

- Control the software

- Understand the numerics: less well understood by the scientific and engineering community.
  - Considering two aspects of algorithm development:
    - Do the numerics accurately capture the physical phenomena?
    - Is the algorithm appropriate for parallel computing?

- Achieve expected behavior: testing, validation, and verification

- Question unexpected behavior
The five rules

– Rolls in CSE: Where to expert?
- Applicationist: Phenomenon, Physics, Mathematics (continuous), Analysis, Verification
- Mathematician: Mathematics (continuous), Numeric (discret), Algorithm, Solution, Analysis
- Computer scientist: Algorithm, Implementation and optimization, Port, Solution

– **Know the physics:** scientists and engineers know their problems well and how to build the mathematical model representing their physical problem

– **Control the software**

– **Understand the numerics:** less well understood by the scientific and engineering community.
  - Considering two aspects of algorithm development:
    - Do the numerics **accurately** capture the physical phenomena?
    - Is the algorithm appropriate for **parallel computing**?

– **Achieve expected behavior:** testing, validation, and verification

– **Question unexpected behavior**

→ **Collaborate** applicationist, mathematician, and computer scientist
Math Modeling and Simulation

– Define the physical problems

– Create a mathematical (PDE) model
  - Systems of PDEs, ODEs, algebraic equations
  - Define Initial and or boundary conditions
  → well-posed problems?

– Create a Discrete (Numerical) Model
  - Discretize the domain → generate the grid
  → obtain discrete model

– Analyse Errors in the discrete system
  - Consistency, stability and convergence analysis

– Solve the discrete system
  - Explicit vs. Implicit for time dependent problems
  - Linear or Nonlinear
Linear systems

- For elliptic problems: Getting Potentials in PIC method

- For semi-implicit method and implicit methods: Stability
- Inner routines for nonlinear problems
Linear system solver

Discretization

\[ \text{PDEs} \quad \Rightarrow \quad Au = f \]
(linear or nonlinear system)

\[
\begin{pmatrix}
  a_{11} & a_{12} & a_{13} & \cdots & a_{1N} \\
  a_{21} & a_{22} & a_{23} & \cdots & a_{2N} \\
  a_{31} & a_{32} & a_{33} & \cdots & a_{3N} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN}
\end{pmatrix}
\begin{pmatrix}
  u_1 \\
  u_2 \\
  u_3 \\
  \vdots \\
  u_N
\end{pmatrix}
=
\begin{pmatrix}
  f_1 \\
  f_2 \\
  f_3 \\
  \vdots \\
  f_N
\end{pmatrix}

\[ A \text{ is invertible} \rightarrow \text{Exists a Unique solution} \]
Basic idea of Multigrid Method

– Motivation: Simple iterative method reduces well a high frequency error and a low frequency error is well approximated by coarser level problem

A Multigrid V-cycle

**Smoothen**

Finest Grid

Restriction
transfer from fine to coarse grid

coarse grid has fewer cells (less work and storage required)

Recursively apply
this idea until we have an easy problem to solve

First Coarse Grid

Prolongation
transfer from coarse to fine grid
Multigrid method: Properties

– Well-known and well-analyzed fast solver and preconditioner
– The required number of iterations is **fixed** for many cases
- For 3-d problem with $n$ DoF for each direction

<table>
<thead>
<tr>
<th>Method</th>
<th>Storage</th>
<th>Flops</th>
</tr>
</thead>
<tbody>
<tr>
<td>GE (banded)</td>
<td>$n^5$</td>
<td>$n^7$</td>
</tr>
<tr>
<td>Gauss-Seidel</td>
<td>$n^3$</td>
<td>$n^5 \log n$</td>
</tr>
<tr>
<td>Optimal SOR</td>
<td>$n^3$</td>
<td>$n^4 \log n$</td>
</tr>
<tr>
<td>CG</td>
<td>$n^3$</td>
<td>$n^{3.5} \log n$</td>
</tr>
<tr>
<td>Full MG</td>
<td>$n^3$</td>
<td>$n^3$</td>
</tr>
</tbody>
</table>

– Smoothing operators: On each level
– Prolongations and restrictions: Intergrid transfer operators
– Exact solver on lowest level: CGM, GMRES, Direct solver, ...
Smoothing operators

- Reduce high frequency error
- Simple and easy to be implemented
- Richardson smoother: Use theoretical analysis
- Jacobi iteration: Simple, but slow.  
  → Does not depend on the number of cores  
  → Good for debugging of parallel code
- Gauss-Seidel: Very good and simple, hard to parallelize  
  - Most preferred smoothing operator
- SOR, SSOR
- Incomplete LU factorization: Very good smoothing property,  
  complex, hard to analyze
- Kaczmarz, AID, etc.
Geometric multigrid method (GMG)

- Separately construct discretization system on each level
- Natural intergrid operators
- Relatively fast $\leftrightarrow$ Need geometric informations
- Top-down approach: From the finest mesh to coarser. Prefered by physicist. The boundary may not match on the coarsest mesh $\rightarrow$ Need a special treatment.
- Down-top approach: From the coarsest mesh to finer. Prefered by numerical analysists. Hard to match complex domains
- According to finite dimensional spaces and discretization
  - conforming or nonconforming
  - finite element, finite difference, or finite volume method
  - cell-centered method or vertex-centered method
Algebraic multigrid method (AMG)

- Black box method
- Define intergrid transfer operator according to the matrix
- More time to generate intergrid transfer operators
- Doesn’t need geometric information
- Can be made as libraries

Examples
- Hypre: LLNL, Solving large, sparse linear systems of equations on massively parallel computers, structured multigrid and element-based algebraic multigrid
- SAMG (Algebraic multigrid methods for systems): Fraunhofer Institute for Algorithms and Scientific Computing
- AMGCL: Generic Algebraic multigrid Hierarchy Builder
Krylov subspace method

Krylov subspace: \( K_m(A; \nu) = \text{span}\{\nu, A\nu, \ldots, A^{m-1}\nu\} \)

- Conjugated gradient method (CGM), GMRES
- Error reduction: Condition number \( \kappa(A) = \frac{\lambda_N}{\lambda_1} \)

\[
\rho = \frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}
\]

<table>
<thead>
<tr>
<th>Levels</th>
<th>DoF</th>
<th>iterations</th>
<th>iteration/(\sqrt{\text{DoF}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>12 097</td>
<td>223</td>
<td>2.03</td>
</tr>
<tr>
<td>7</td>
<td>48 769</td>
<td>435</td>
<td>1.97</td>
</tr>
<tr>
<td>8</td>
<td>195 841</td>
<td>839</td>
<td>1.90</td>
</tr>
<tr>
<td>9</td>
<td>784 897</td>
<td>1 636</td>
<td>1.84</td>
</tr>
<tr>
<td>10</td>
<td>3 142 657</td>
<td>3 169</td>
<td>1.79</td>
</tr>
<tr>
<td>11</td>
<td>12 576 769</td>
<td>5 001</td>
<td></td>
</tr>
</tbody>
</table>
Preconditioner

– The preconditioned problem

\[
\hat{A}M\hat{x} = \hat{M}b, \quad \text{or} \quad AM\hat{M}^{-1}\hat{x} = b, \quad \text{or} \quad M_LAM_R\hat{M}^{-1}\hat{x} = M_Lb.
\]

– \(\hat{A}\) has a small condition number: \(M^{-1} \approx A\)
– Easily solve \(M^{-1}y = z\)
– Preconditioner does not affect on truncation error
   → Only effect on convergence rate to solve the system

– The multigrid method
– Domain decomposition method
MPI: Message Passing interface

- Standardized and portable message-passing system
- A language-independent communication protocol
- Begin summer of 1991 and released V1.0 June 1994

- Launch one process per core, on one node or on many
- Pass messages among processes without concern for location
- Maybe create different communicators intra-node vs. inter-node
- Ignore the potential for any memory to be shared
Parallelization issues on MPI

- Lower levels: needs more data communication cost in comparison to computing cost → Bottleneck on the parallel computers
- Use V-cycle scheme as a solver and/or as a preconditioner
- Gauss-Seidel smoother: Hard to parallelization
  → Localize:
    - Perform the Gauss-Seidel iteration exclusively on each core
    - No data communication between cores in one Gauss-Seidel cycle

**Gathering data**

- Reduce the number of execution cores on a certain coarser level
  → Use only one core
- Gather data to one core, solve, and scatter to all core
  → Only one core is busy and others idle
- Gather data on each core and solve on every core
  → Don’t need scattering step
- **Use** `MPI_Allreduce`: Combine `MPI_Reduce` and `MPI_Bcast`
  → Better performance depending on the MPI implementation
Gathering data algorithm

V-cycle Multigrid Method

- Highest Level
  - Multi-core levels
    - Restriction
    - Pre-smoothing
  - Gathering level
    - Single-core levels
      - Post-smoothing
    - Lowest level solver
OpenMP

– API that supports multi-platform shared memory multiprocessing programming
– Use a portable scalable model → a simple flexible interface for developing parallel applications
– By OpenMP Architecture Review Board (ARB)
  - OpenMP 1.0 for Fortran (1997.10) for C/C++ (1998.10)
  - V2.5 combined C/C++/Fortran in 2005
  - V4.0: Released in July 2013

– Have each process fork one thread (or maybe more) per core
– Share data using shared memory
– Can’t share data with a different processor (node) (except maybe via file I/O)
OpenMP/MPI hybridization

– Hybridization: Each MPI task (process) to launch multiple OpenMP threads that can share local memory
- Can’t share data with a different processor (node) (except maybe via file I/O)
OpenMP/MPI hybrid: Expectations

– Pure MPI: Long MPI initialization time on a large number of MPI tasks (> 5000)
– OpenMP: Need thread launching time
  ↔ Compare with MPI communication time

– Hybridization with one thread per core:
  → Use same number of cores with smaller number of MPI tasks
  - Better scaling property on the massively parallel computers

– Localized Gauss-Seidel smoother: Don’t wait to update values which handles by other threads
  → Racing condition, might not be same numerical results
Current and future system: Cluster

– Collections of workstations/PCs connected by a local network
– 1994: The first Beowolf cluster, ICASE
– The speed of the network is crucial
– Multi-threaded, fine-grained concurrency of 10- to 100-way concurrency per core (computational accelerator) 4–8 processors per node
– ccNUMA memory access

– Major system type: Each node CPU + GPU (or MIC)
– 100 million to 1 billion cores with clock rates of 1 to 2 GHz
– Active power management
– 3D packaging of dies for stacks of four to ten dies, each including DRAM, cores, and networking
– **Intel Xeon E5**: Ivy Bridge-EP cores, 2200MHz, Tianhe-2 (1, 2692, 12C), Cray CS-Storm(10,2660v2, 10C) - Sandy Bridge-EP cores, 2600MHz, Piz Daint (6, 2670,8C), Stampede (7,2680,8C)

– AMD Opteron 6200 (Interlagos): multi-chip module consisting of two hex-core dies, 2.2 GHz, Titan(2,6274,16C)

– IBM Blue Gene/Q PowerBQC: High energy efficiency design. 1.6GHz, 5D Torus network - Sequoia (3,1,572,865 cores), Mira(5,786,432), JUQUEEN(8, 458,752), Vulcan(9,393,216)

– IBM Power7 chips: multiple cores (4, 6 or 8) and 4 simultaneous multithreading (SMT), 3.0 – 4.25 GHz

– Fujitsu SPARC64 VIIIfx: 8C, 2GHz, K computer (4)
Computational accelerators: simple processors

– Intel Many Integrated Core Architecture (MIC):
  - Manycore extended x86/x64 design
  - Xeon Phi: a PCI card with 60 (61) in-order cores at up to 1.2GHz with 4 threads per core
  - OpenMP, OpenCL, Cilk/Cilk Plus, ....
  - Tianhe-2 (1), Stampede (7), cascade(18), ...

– NVIDIA Graphical processing units (GPU):
  - Large number of relatively simple processors,
  - NVIDIA Tesla K20X: 2688 cores, 1.31 Tflops/s CUDA, OpenCL
  - Titan(2), PizDaint (6), Cray CS-Storm (10), ...

– General computational accelerators: PetaPath, Cell processor

– FPGA-based accelerators: An array of logic gates that can be hardware-programmed to fulfil user-specific tasks,
  Hardware description language (HDL, VHDL or Verilog)
  - Convey, Kuberre, SRC
Model problem

- Poisson problem on a regular hexagonal domain with Dirichlet boundary condition

\[
\begin{cases}
-\nabla \cdot \nabla u = f, & \text{in } \Omega \\
u = 0, & \text{on } \partial \Omega
\end{cases}
\]

- Computing Potentials on each time step
- Finite element method: For any test functions \( v \)

\[
\int_{\Omega} \nabla u \cdot \nabla v dx = \int_{\Omega} f v dx.
\]
Discretization

– Triangulation with regular triangles
– Boundary nodes: No degree of freedom. Ghost nodes.

– Use Linear finite element method or finite volume method
Parallelization: Division

– Divide a regular hexagonal domain with regular triangular sub-domains
– Only limited number of cores: 1, 6, 24, 96, 384, ...
Parallelization: Numbering

– Sub-domain numbering: From the origin and counterclockwise direction
Parallelization: Subdomains

- Determine where the boundary nodes of the sub-domain are included.
- Consisted by Real (●) Ghost (○) nodes.
- Classify three types of sub-domains
  
  Type I: 0, 6, 9, 12, 15, 18, 21, 24, ...
  
  Type II: 1, 2, 3, 4, 5, 8, 11, ...
  
  Type III: 7, 10, 13, 16, 19, 22, 25, ...
Data communication: five steps

Step1: Radical direction (a)
Step2: Counterclockwise rotational (b)
Step3: Clockwise rotational (c)
Step4: Radical direction (a)
Step5: Mixed Communication (d)
High Level Support Team (HLST)

– Started as a support unit under EFDA at 2009
– Continue under EUROfusion consortium

– Ensure to use the dedicated HPC to fusion researchers in EU
  - HPCFF
  - Helios

– Support to scientists from all Research Units of the EUROfusion consortium for the development and optimization of codes
  - Prepare supporting new computers
– Reason: Architecture of High Performance computer and its programming are too complex to be handled by Plasma Physician alone
  → Computational Science and Engineering (CSE)
Members

– HPC experts with a background in developing large scientific applications and particular expertise in numerical algorithms

– Core team members: 6 persons, IPP, Garching, Germany
  - 4 physicist (former 3)
  - 1 computer scientist (former 2)
  - 1 mathematician

– Staff members: Other European Fusion institutes in EU
  - 50% HLST and 50% Institute
  - Maximum 4 staff members planned

– Projects: From almost all European Fusion research Institutes
  2009-2010 (6), 2010-2011 (13), 2011-2012 (12),
  2012-2013 (11), 2013 (10), 2014 (14), 2015 (8)
Tasks

– **Parallelise** codes using e.g. OpenMP and/or MPI standards for massively parallel computers
– **Improve the performance** of existing parallel codes both at the single node and inter node levels
– Support the **transfer** of codes to new multiprocessors architectures
– Choose and adapt **algorithms and mathematical library routines** to improve applications for the targeted computer architectures
– **Give feedback** to the community based on experience gained from specific project work
– **Provide guidance** for young scientists on available training activities in HPC and towards upcoming new computer architectures
– **Provide consultancy** to scientists within the Research Units working on HPC
– **Exploit developments** made by the WPCD, especially in the field of standards, graphical user interfaces, common data bases and parallel visualization, for the benefit of the IFERC-CSC users.
Helios: 4410 Bullx B510 Blades, 70,000 cores

– IFERC: The International Fusion Energy Centre at Rokkasho, Japan
  - EU(F4E)—Japan Broader Approach collaboration

– The Computational Simulation Centre (CSC): To exploit large-scale
  and high performance fusion simulations

– Optimally suited for the fusion scientists’ simulation programs
– 1.237 Pflops max and 1.52 Pflops peak performance (No. 38 in
  Top500, Nov. 2014)
– Two Intel Sandy-Bridge EP 2.7 GHz per node, Infiniband
  Interconnection, 180 MIC node added

K. S. Kang (IPP)
Algorithmic scaling

- The required number of computations to solve system with size $N$

<table>
<thead>
<tr>
<th>Method</th>
<th>Storage</th>
<th>Flops</th>
</tr>
</thead>
<tbody>
<tr>
<td>GE(banded)</td>
<td>$n^5$</td>
<td>$n^7$</td>
</tr>
<tr>
<td>Gauss-Seidel</td>
<td>$n^3$</td>
<td>$n^5 \log n$</td>
</tr>
<tr>
<td>Optimal SOR</td>
<td>$n^3$</td>
<td>$n^4 \log n$</td>
</tr>
<tr>
<td>CG</td>
<td>$n^3$</td>
<td>$n^{3.5} \log n$</td>
</tr>
<tr>
<td>Full MG</td>
<td>$n^3$</td>
<td>$n^3$</td>
</tr>
</tbody>
</table>

To solve the Poisson problem in 3D ($N = n^3$)

- Required flops is not enough on massively parallel computer
Strong and weak scaling:

- Strong scaling: Execution time decreases in inverse proportion to the number of processors fixed size problem over all.
- Weak scaling: Execution time remains constant, as problem size and processor number are increased in proportion fixed size problem per processor (Gustafson scaling).
- More important on massively parallel computer.

![Diagram showing strong and weak scaling](image)
Matrix-Vector multiplication

– Basic routine of the iterative methods

execution time (log scale)

0.1 0.01 0.001 0.0001 0.00001

number of cores (log scale)

6 24 96 384 1536 6144 24576

527K DoF/core
132K DoF/core
33K DoF/core
8.5K DoF/core
2.2K DoF/core

– Almost perfect scaling property
CGM

K. S. Kang (IPP) Parallel MG GIST HPC 38 / 48
With and Without gathering data

- $2.2\,\text{K DoF/core}$
- $8.5\,\text{K DoF/core}$
- $33\,\text{K DoF/core}$
- $132\,\text{K DoF/core}$

solution time (log scale)

number of cores (log scale)
With gathering data

solution time (log scale)

(a) Level 12 (384 cores)
(b) Level 13 (1536 cores)
(c) Level 14 (6144 cores)

Lowest gathering level

═・・・ second  ○○○ third
Numerical results: Strong scaling

- Black: PCGMG
- Red: MGGS

- : 3.1M DoF
- : 12.5M DoF
- : 50M DoF
- : 201M DoF
- : 805M DoF
Numerical results: Weak scaling

Black: PCGMG
Red: MGGS

- 2.2K DoF
- 8.5K DoF
- 33.4K DoF
- 132K DoF
- 527K DoF
- 2.1M DoF

per core
OpenMP effects

(a) 96 MPI tasks

MG with GS as a preconditioner of PCG in Black, Matrix-Vector multiplication in Red

(a) ···: ideal, —: 3M DoF, o o o: 12.5M DoF, + + +: 50M DoF
(b) ···: ideal, —: 12.5M DoF, o o o: 50M DoF, + + +: 200M DoF
Strong scaling: 3.1M DoF

—–: pure MPI, the best in red
Strong scaling: 12M DoF

---

+ pure MPI, the best in red

---

Parallel MG

K. S. Kang (IPP)

GIST HPC
Weak scaling: Hybridization

The best (black)
Pure MPI (red)
2.2K DoF (solid)
8.5K DoF (○)
3.1M DoF (●)
13M DoF (×)
50M DoF (⋄)
per core
Conclusions

– Sophisticated communication pattern:
  → Very good performance for the matrix-vector multiplication and CGM

– The multigrid algorithm: Very good semi-weak scaling property up to 24,576 cores for the large number of DoF per core

– Small number of DoF per core and large number of MPI tasks
  → Improved the performance by using hybrid OpenMP/MPI
Acknowledgment

– HLST core team members (Leader: R. Hatzky)
– B. Scott (project coordinator)
– D. Tskhakaya (project coordinator)

Thank you for your attention