# 超新星爆発シミュレーションのための
# 球対称ニュートリノ輻射流体計算のGPUによる高速化
# (II)

Hideo Matsufuru

*High Energy Accelerator Research Organization (KEK)*

Kosuke Sumiyoshi

*National Institute of Technology, Numazu College*

Annual Meeting of Physical Society of Japan

23 March 2017, Tokyo University of Science, Noda, Japan

# Introduction

Core collapsed supernovae

- Large scale numerical simulation is essential to understand explosion mechanism
  - Hydrodynamics
  - Boltzmann equation for neutrino transport
  - General relativity
  - Equation of state of dense matter, neutrino reactions
- Fully 6D simulations are currently restrictive
  - Dimensionality plays an essential role for explosion
  - Approximations are often used for 2D/3D systematics
  - Acceleration of full 2D/3D simulations are waited
- Spherically symmetric system
  - Basis for 2D/3D simulations and observations
  - Systematic survey of massive stars is necessary
  - $1^{st}$ principle calculation (Full GR + Hydro + Boltzmann)

# Introduction

High performance computing

- Two trends of architecture
- Massively parallel supercomputers
  - K-computer → Post-K (2021)
  - Intel Xeon Phi
  - Preparation for the next generation SC is underway
- Arithmetic accelerators
  - GPUs, Pezy-SC
  - CPUs + devices: Heterogeneous architecture
  - Currently not widely used in SN simulations in spite of large potential

*This work: for establishing techniques to exploit heterogeneous architectures for supernova simulations*

- Application to spherically symmetric system as a testbed

# Implementation

- Numerical setup
  - GR Lagrangian hydrodynamics + $S_N$ + Implicit scheme

    S. Yamada, ApJ 475 (1997) 720, A&A 344 (1999) 533
  - Ar every step of time evolution, solve a linear equation system
  - BiCGStab algorithm for a block tridiagonal matrix

$$M = \begin{pmatrix} B_1 & C_1 & 0 & & \dots & \\ A_2 & B_2 & C_2 & 0 & & \\ 0 & A_3 & B_3 & C_3 & & \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ & & 0 & A_{n-1} & B_{n-1} & C_{n-1} \\ 0 & \dots & & 0 & A_n & B_n \end{pmatrix}$$

  - Rank of each block matrix: $N_{max} = N_{E_\nu} \cdot N_{\rm ang} \cdot N_\nu + N_{\rm hyd}$
  - Weighted Jacobi preconditioner

    A. Imakura et al. JSIAM Letter 4 (2012) 41

$$x_{k+1} = \omega \left[ -M_D^{-1}(M - M_D)x_k + M_D^{-1}b \right]$$

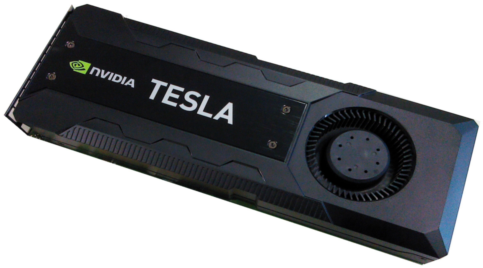
  - This linear equation solver is the first target of offloading

# Implementation

- Offloading scheme
  - Base code: Fortran, MPI parallelized
  - C-code as intermediate code (for later convenience)
  - Device code is implemented with OpenACC
    - Directive-based framework for offloading
    - Portable, compatible with OpenMP 4.0
  - Multi-GPU: each MPI process handles one GPU device
  - Communication via host process
- Two implementations
  - "Native" code
    - Tuned using OpenACC directives
    - Simple assignement of tasks to threads: each thread compute one row
  - cuBLAS code
    - Well-tuned BLAS library provided by NVIDIA
    - CUDA stream for asynchronous execution (in units of block matrix)

# Machines

- Performance is measured on two machines at KEK
  - All arithmetic operations are in double precision
  - Compiler: PGI + OpenMPI + CUDA environment

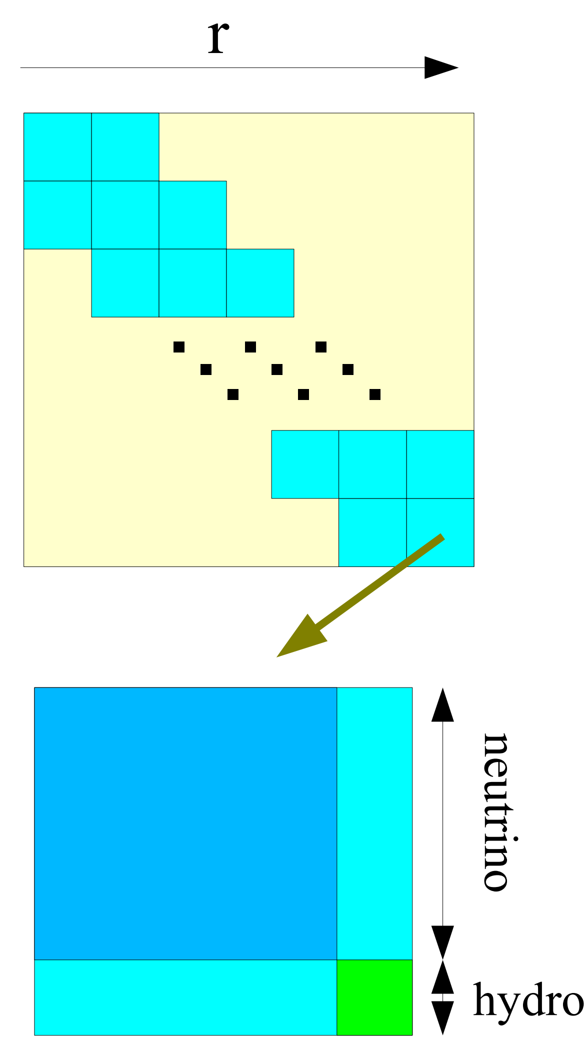



| | Xeon + Kepler | Power8 + Pascal |
|---|---|---|
| Host processor | Intel Xeon Haswell (6 core) x2 | IBM Power8 (10 core) x2 |
| Peak/core (double) | 44.8 GFlops/core | 22.9 GFlops/core |
| GPU | NVIDIA K40 x2 | NVIDIA P100 x4 |
| #core/device (double) | 960 | 1792 |
| Peak/device (double) | **1430 GFlops** | **4700 GFlops** |
| Memory size/device | **12** | **16** |
| Memory B/W | **288 GB/s** | **720 GB/s** |
| bus | PCIe Gen2 x16 | NVLink (40GB/s x2) |

# Problem setup

- Linear equation
  - At each spatial (radial) point r,
    - Neutrino dof.: energy, angle, species
    - Hydrodynamical dof.: velocty etc. ($N_{hyd}$=11)
    - Block tridiagonal matrix, each block is dense

    → matrix rank: $N_{max} = N_E * N_{ang} * N_\nu + N_{hyd}$
  - $N_r$ = 256 → increased to 512, 1024 (preferable)
  - Required memory size = $N_{max}^2 * 4 * N_r * 8$ Byte

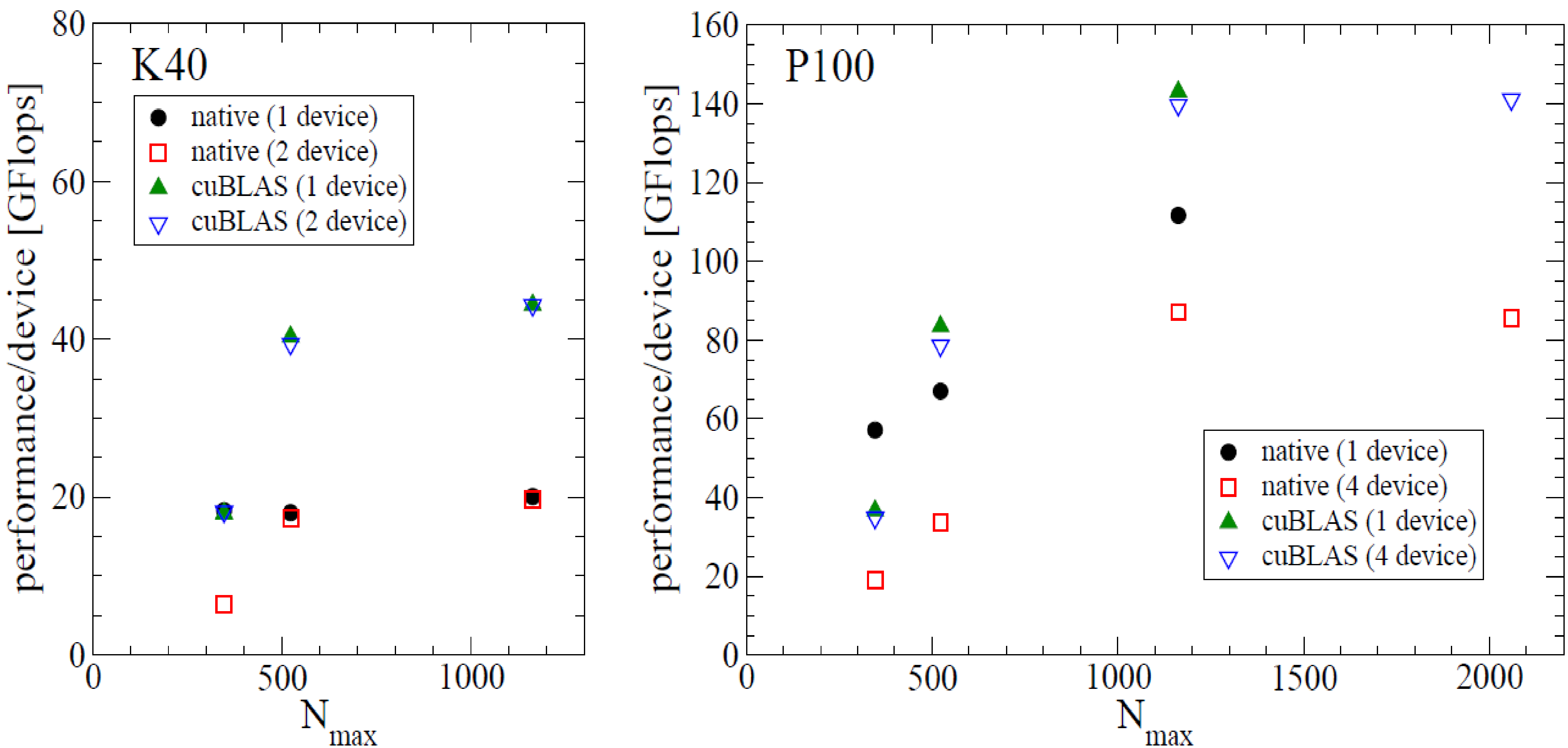



| | set-1 | set-2 | **set-3** | set-4 |
|---|---|---|---|---|
| $N_r$ | 256 | 256 | 256 | 256 |
| $N_E$ | 14 | 16 | **24** | 32 |
| $N_{ang}$ | 6 | 8 | **12** | 16 |
| $N_\nu$ | 4 | 4 | **4** | 4 |
| $N_{max}$ | 347 | 523 | 1163 | 2059 |
| memory | 1 GB | 2 GB | **9.3 GB** | 32 GB |

Practical choice of $N_{max}$

# Performance result

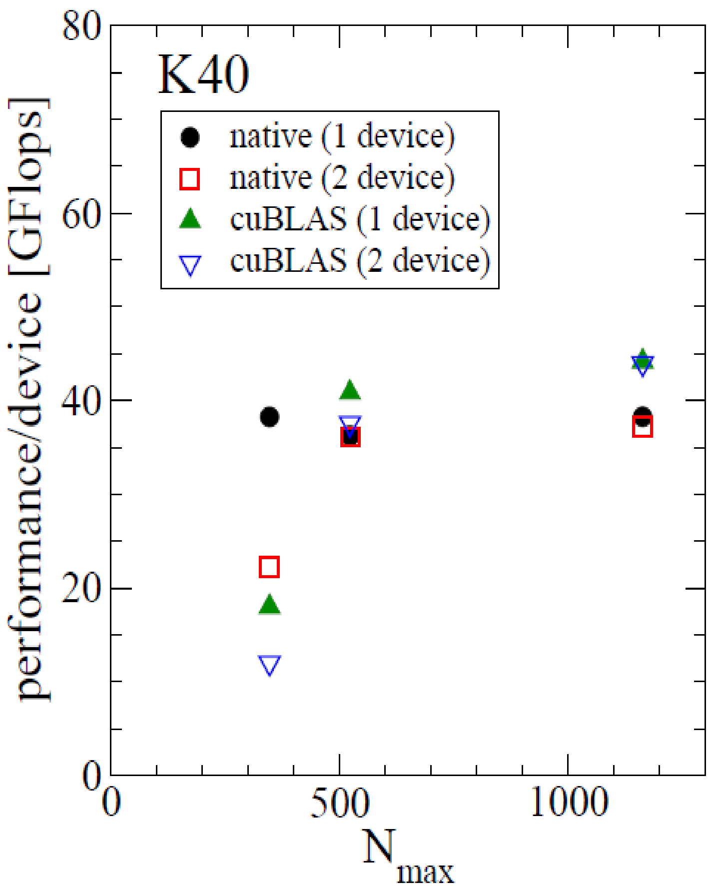
- $M_D^{-1}$ : inverse of diagonal blocks (no communication)

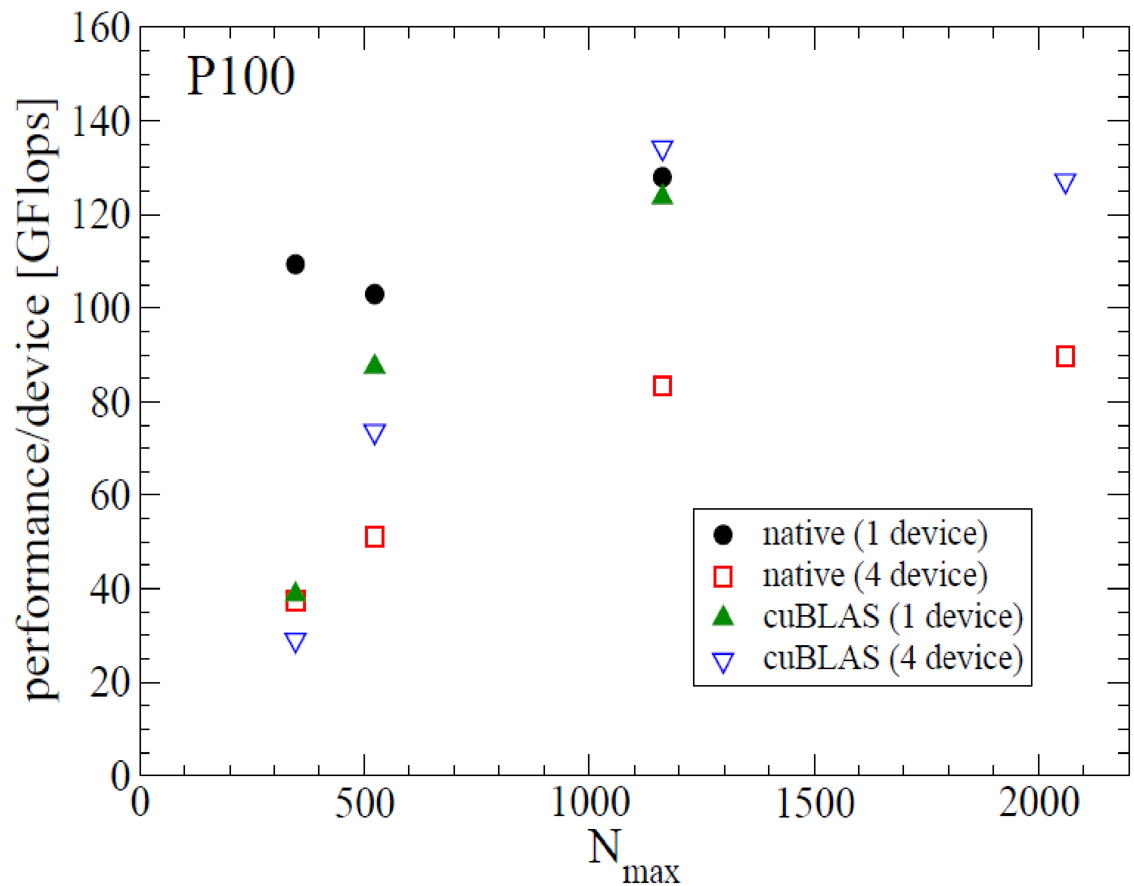



  - cuBLAS code tends to be faster at large $N_{max}$ region (~x1.5)
  - On Pascal, ~140 Gflops/device (3% of peak)

# Performance result

- $M - M_D$ : subdiagonal blocks (with communication)




- Communication overhead is small (~$1/N_{max}$*[$N_r/N_{dev}$])

# Acceleration of simulation

- How much is simulation accelerated ?
- Time consuming parts in evolution step
  - Dominant: matrix inversion
  - Subdominant: neutrino reaction (1/20 of matrix inv.)
  - Other parts are (currently) negligible

Result: next page

- GPUs largely accelerate the linear solver
- Now dominant operations are:
  - Preparation of matrix (determination of weight parameter)
  - Neutrino reaction

→ Next targets of offloading

# Acceleration of simulation

- Speed up
  - Elapsed time [sec] is measumred for original and offloaded codes
  - Xeon + Kepler: Nrmax=256, Nang=6, Ne=14

| **Xeon/Kepler (Set-1)** | Np = 8 | Np = 1 | GPU (cuBLAS) |
|---|---|---|---|
| collision | 4.9 | 40.5 | |
| Matrix total | 115.5 | 911.4 | 46.2 |
| setup | 5.1 | 40.2 | 40.8 |
| Inversion (11 iter) | 110.4 | 871.2 | **5.4** |

  - Power8 + Pascal (x4): Nrmax=256, Nang=8, Ne=16

| **Power/Pascal (Set-2)** | Np = 16 | Np = 4 | GPU (cuBLAS) |
|---|---|---|---|
| collision | 8.6 | 35.1 | |
| Matrix total | 168.0 | 622.5 | 45.9 |
| setup | 11.8 | 43.7 | 45.0 |
| Inversion (7 iter) | 156.2 | 578.8 | **0.9** |

# Conclusion/outlook

- Conclusion
  - Acceleration of 1D neutrino-radiation hydrodynamics code by GPUs
  - Efficient development by using OpenACC (+cuBLAS)
    - cuBLAS becomes more efficient for larger block matrices, while OpenACC also works well
  - Large speed up for matrix inversion required in implicit scheme
    - GPU performance is not well exploited, but already enough fast
    - Other parts are to be accelerated urgently
    - Simulations with $N_r$=1024, $N_{ang}$=12, $N_E$=24 becomes practical

  → Application to systematic 1D simulations
- Outlook
  - Application to 2D, 3D code
  - Further optimization and improvement of algorithms
  - Pezy-SC