計算数理工学論文集 Vol. 15 (2015 年 12 月), 論文 No. 11-151204 JASCOME

メモリアクセスを考慮した輻射輸送方程式の並列計算の検討

A STUDY ON LARGE SCALE PARALLEL COMPUTATION OF THE RADIATIVE TRANSPORT EQUATION WITH MEMORY ACCESS ANALYSIS

藤原宏志 [1)]

Hiroshi FUJIWARA

1) 京都大学大学院 情報学研究科 (〒 606-8501 京都市左京区吉田本町, E-mail: fujiwara@acs.i.kyoto-u.ac.jp)

We propose hybrid parallel computation for the radiative transport equation (RTE). RTE is an integro-differential equation and its discretization scheme contains several manners of operations and memory access. Graphics processing unit (GPU) is applied to compute the scattering integral in RTE, and data transfer time is hidden by using the software pipelining technique. We also show some numerical examples and discuss its efficiency in consideration of memory access.

Key Words: Radiative Transport Equation, GPGPU, Heterogeneous Computing, Hybrid Parallelization, Memory Bandwidth

1. 緒言

近年，脳科学の分野では，近赤外光による生体の非侵襲モニタリング法の開発が進められている [(1, 2)]．そこでは生体中での近赤外光伝播の数理モデルに輻射輸送方程式 (RTE; Radiative Transport Equation) がもちいられ，その高速・高精度な数値計算法の開発が嘱望されている．本研究は RTE の数値計算の高速化のために，近年一般的となっているマルチコア CPU と GPU を有するノードからなる PC クラスタ環境において，RTE に現れる作用素ごとに離散化の特性を考慮したヘテロジニアス計算の有効性を，メモリアクセスの検討も含めて示す．

定常状態に対応する 3 次元輻射輸送方程式の境界値問題は粒子密度 $I = I(x,\xi), x \in \Omega \subset \mathbb{R}^3, \xi \in S^2 = \{\xi \in \mathbb{R}^3; |\xi| = 1\}$ を未知函数として

$$-\xi \cdot \nabla_x I - (\mu_\mathrm{s} + \mu_\mathrm{a})I + \mu_\mathrm{s} \int_{S^2} p(\xi,\xi')I(x,\xi')d\sigma_{\xi'} = 0, \quad \text{in } X = \Omega \times S^2, \tag{1a}$$

$$I(x,\xi) = I_1(x,\xi), \qquad \text{on } \Gamma_- \tag{1b}$$

で与えられる．ここで $\nabla_x I = (\partial I/\partial x_1, \partial I/\partial x_2, \partial I/\partial x_3)$, $d\sigma_{\xi'}$ は S^2 の面積要素，$n(x)$ を $\partial\Omega$ の外向き単位法線として $\Gamma_- = \{(x,\xi)\,;\, x \in \partial\Omega, n(x)\cdot\xi < 0\}$ である．また $\mu_\mathrm{s}, \mu_\mathrm{a}, p(\xi,\xi')$ は散乱係数，吸収係数，散乱位相函数とよばれる．この 3 次元の RTE の直接離散化は本質的に 5 次元の大規模問題となる．そのため一般的な計算環境での数値的取り扱いは Monte Carlo 法が主流であり [(3)]，画像描画用プロセサ (GPU; Graphics Processing Unit) による高速化が報告されている [(4)]．

2015 年 9 月 20 日受付，2015 年 10 月 30 日受理

一方，本研究の特徴は，適当な条件下で収束性を有する RTE の直接離散スキームを対象とし，RTE に現れる積分および微分作用素の離散化の性質の違いに着目してそれぞれを GPU と CPU で処理することで高速化を図ることにある．RTE は微分積分方程式であり，積分および微分作用素を標準的な数値積分則と差分法で離散化する場合，得られる式の性質は大きく異なる．特に離散化数を一定とするときの演算とメモリアクセスに注目すると，数値積分は空間方向の 1 点 $x \in \Omega$ に対して $\xi \in S^2$ の方向の離散化変数全体にわたるアクセスと乗加算が必要となる一方，差分は数個のデータの演算で実現され，メモリアクセス回数も少ない．本研究ではこの違いを考慮し，散乱積分をメモリ局所的なデータの高速計算に適した GPU で計算し，大域的なメモリアクセスを含む他の処理を CPU でおこなう．

これら CPU と GPU は互いに独立したメモリを有しており，それらメモリ間のデータ転送速度は演算速度に比して遅く，高速化において注意を要する．そこでソフトウエアパイプライン手法により転送時間の隠蔽を試みる．また前述のとおり 3 次元の RTE は大規模問題となるため MPI でのメモリ分散並列化をおこなうとプロセス間通信が生じるが，これも同様の手法により隠蔽する．したがって提案手法はデータを共有すべき並列と分散すべき並列からなるハイブリッド並列となる．一方でこのような CPU と GPU を混在させる計算におけるメモリアクセスやデータ転送についての特性は十分に知られておらず [(5)]，本研究では，計算事例においてメモリアクセスを調べ，提案手法と実装の効率性を論じる．

以下，本論文では CPU からアクセス可能なメインメモリ

(ホストメモリ) を単にメモリ，GPU デバイスの演算器とメモリをともに GPU と記す．また本研究では NVIDIA 社製の CUDA [6] により GPU デバイスを利用する．

2. 定常輻射輸送方程式の離散化と Gauss-Seidel 法

領域 Ω を $\mathbb{R}^3$ の直方体 $(0,L_1)\times(0,L_2)\times(0,L_3)$, N_1,N_2,N_3 を正整数とし，$\Delta x_i = L_i/N_i$, $x_{ijl} = (i\Delta x_1, j\Delta x_2, l\Delta x_3)$ とする．また速度方向の離散化のため S^2 上に $\{\xi_\nu\}_{\nu=1}^M$ をとり，$I(x_{ijl},\xi_\nu)$ の相当値を $I_{ijl,\nu}$ として次を考える [7]．

$$
\begin{aligned}
A_\Delta I_{ijl,\nu} = &-\xi_{\nu,1}\frac{I_{i+1,j,l,\nu}-I_{i-1,j,l,\nu}}{2\Delta x_1}\\
&+|\xi_{\nu,1}|\frac{I_{i+1,j,l,\nu}-2I_{i,j,l,\nu}+I_{i-1,j,l,\nu}}{2\Delta x_1}\\
&-\xi_{\nu,2}\frac{I_{i,j+1,l,\nu}-I_{i,j-1,l,\nu}}{2\Delta x_2}\\
&+|\xi_{\nu,2}|\frac{I_{i,j+1,l,\nu}-2I_{i,j,l,\nu}+I_{i,j-1,l,\nu}}{2\Delta x_2}\\
&-\xi_{\nu,3}\frac{I_{i,j,l+1,\nu}-I_{i,j,l-1,\nu}}{2\Delta x_3}\\
&+|\xi_{\nu,3}|\frac{I_{i,j,l+1,\nu}-2I_{i,j,l,\nu}+I_{i,j,l-1,\nu}}{2\Delta x_3},\\
\Sigma_\Delta I_{ijl,\nu} = &\big(\mu_{\rm s}(x_{ijl})+\mu_{\rm a}(x_{ijl})\big)I_{ijl,\nu},\\
K_\Delta I_{ijl,\nu} = &\,\mu_{\rm s}(x_{ijl})\sum_{\mu=1}^{M} w_\nu p(\xi_\nu,\xi_\mu)I_{ijl,\mu}.
\end{aligned}
$$

$A_\Delta I_{ijl,\nu}$ では $\xi_\nu = (\xi_{\nu,1},\xi_{\nu,2},\xi_{\nu,3})$ であり，これは $-\xi\cdot\nabla_x I$ の (x_{ijl},ξ_ν) での差分近似を与える．また $K_\Delta I_{ijl,\nu}$ は散乱積分の離散化である．これらの記号のもとで，(1) の離散問題として $\{I_{ijl,\nu}\}$ を未知数とする次の連立方程式を得る．

$$
\begin{aligned}
(A_\Delta-\Sigma_\Delta+K_\Delta)I_{ijl,\nu} &= 0, && (x_{ijl},\xi_\nu)\in X,\\
I_{ijl,\nu} &= I_1(x_{ijl},\xi_\nu), && (x_{ijl},\xi_\nu)\in\Gamma_-.
\end{aligned}
$$

この係数行列は適当な条件下で優対角行列となるため Gauss-Seidel 法が有効であり [8]，以下ではその高速化を論じる．

3. 行列乗算による散乱積分の高速計算

RTE に現れる $K_\Delta I_{ijl,\nu}$ の高速計算には CPU のキャッシュメモリを活用した行列乗算の適用の有効性が示されているが [8]，本研究では GPU の高並列性に着目して高速化を図る．そこで本節では CPU と GPU のそれぞれでの行列乗算の計算時間の例を示す．

行列演算は多くの数値計算に現れることから BLAS [9] としてインターフェースが整理され，その実装として CPU では IMKL [10]，GPU では cuBLAS [11] などが高速な実装としてプロセサベンダから提供されている．$K_\Delta I_{ijl,\nu}$ には実数成分の密行列が現れ，倍精度型数の乗算である DGEMM をもちいる．

GPU での時間測定ではメモリと GPU 間の転送時間も調べるため，Fig. 1 に示すプログラムをもちいた．プログラム中，`cudaThreadSynchronize()` は，それ以前に GPU で起動されたすべての処理の終了を待ち，CPU と GPU の実行を同期させる CUDA の API であり，`CPU_TIME()` は CPU 時間を取得するものである．CUDA で提供される行列乗算 `cublasDgemm()` は，その処理の終了を待たずに CPU 側のプログラムの実行が進む非同期 API であることに注意する．すなわち `cublasDgemm()` が起動されれば，その結果が GPU メモリに書き込まれる前に CPU で次の処理が実行される可能性がある．したがって Fig. 1 で `cudaThreadSynchronize()` の同期がなければ，t_2-t_1 は `cublasDgemm()` の起動に要する時間に相当し，その実行に要する時間とならない．一方，データ転送をおこなう `cublasSetMatrix()`, `cublasGetMatrix()` はいずれも同期 API であり，明示的な同期は不要である．

測定結果を Table 1 に示す．ベンチマークでは Table 2 の環境を利用し，1952 次の正方行列 A と，1952 行 15000 列の長方行列 B に対し，$C = A\times B$ に要する時間を計測した．行列 A は散乱位相函数と積分則の重みに対応し，計算中に更新されないと想定して，Gauss-Seidel 法の反復で更新される未知数に相当する B,C の転送時間のみを測定した．表中，中列が演算に要する時間を示す．またこの行列乗算は 5.72×10^{10} 回の乗加算 (Fused Multiply-Add) で実現されることから，1 秒間に処理可能な FMA の回数を算出し，右列に示した．中段はメモリと GPU 間の転送速度を含めた時間 (t_3-t_0) を，最下段が転送速度を除く GPU での演算時間 (t_2-t_1) を示す．この結果から，GPU の計算ではメモリとのデータ転送が 42% (68 ミリ秒) を占めていることがわかる．また転送を含めない行列乗算のみの演算は，GPU では CPU の 22.5 倍高速であり，RTE の計算において GPU の利用が有効であること，およびデータ転送速度を考慮した高速化の必要性がわかる．

```
t0 = CPU_TIME();

行列 B をメモリから GPU へ転送
(cublasSetMatrix)

t1 = CPU_TIME();

cublasDgemm(handle,
        CUBLAS_OP_N, CUBLAS_OP_N,
        m, l, m, &a, A, m, B, m, &b, C, m);

cudaThreadSynchronize();

t2 = CPU_TIME();

行列 C を GPU からメモリに転送
(cublasGetMatrix)

t3 = CPU_TIME();
```

Fig. 1: Benchmark Program for Matrix Multiplication.

4. メモリと GPU 間の転送時間の削減

GPU での計算では，Table 1 に示すとおりデータ転送の割合が大きいことがわかる．そこでこの転送を，Gaussi-Seidel

Table 1: Computation Times of DGEMM $C = AB$. Sizes of A and B are 1952×1952 and 1952×15000 Respectively.

	Time [msec]	FMA [ops/sec]
IMKL	2113	27.1×10^9
cuBLAS, $t_3 - t_0$	162	—
cuBLAS, $t_2 - t_1$	94	607×10^9

Table 2: Hardware Specifications in Numerical Experiments.

CPU	Core i7-4770 (3.4GHz, 4 Cores)
Main Memory	32GB DDR3 (PC3-12800)
NIC	Intel GbE I217-V (OnBoard)
Operating System	Linux CenOS 6.5
C/C++	gcc-4.4.7-4.el6.x86_64
MPI	mpich2-1.2.1-2.3.el6.x86_64
Intel MKL	11.1.0.080
GPU	NVIDIA GTX TITAN (GK110, 2688 Cores, 837MHz)
GPU Memory	6GB GDDR5
CUDA	5.5.22
HUB	DELL PowerConnect2724

法に現れる他の演算と同時に実行することで全体の計算の高速化を図る．

RTE の Gauss-Seidel 法では，$A_\Delta, \Sigma_\Delta, K_\Delta$ に相当する計算の後に未知数 $I_{ijl,\nu}$ を更新する．前述のとおり K_Δ の計算に GPU をもちいるには，未知数 $I_{ijl,\nu}$ を添字 ν について連続するメモリ領域に保持することが望ましい．その場合，A_Δ の計算ではメモリ上で離れたアドレスへのアクセスを要するため GPU での計算には適さない．そこで Σ_Δ および値の更新を合わせて CPU で実行するものとする．

大規模問題である 3D RTE の計算を MPI で並列化すると，プロセス間通信によるデータ交換も生じる．これも含めて，RTE に対する Gauss-Seidel 法の計算の 1 回の反復は次の 5 つの処理 (ステージ) で構成される．

(S1) メモリから GPU へのデータ転送

(S2) GPU での K_Δ 相当の計算

(S3) GPU からメモリへのデータ転送

(S4) CPU での A_Δ, Σ_Δ 相当の計算と未知数の更新

(S5) MPI でのプロセス間のデータ交換

並列計算にブロック Gauss-Seidel 法をもちいることにすると，未知数 $\{I_{ijl,\nu}\}$ を差分や積分則による離散化，すなわち連立方程式での関係を考慮して適当に N 個のブロックに分割し，(S1)–(S5) の手順をブロック毎に実行する．この手順を処理対象のブロック番号を i として Fig. 2 に示す．

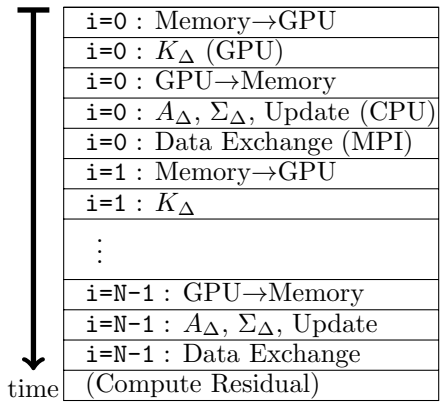


Fig. 2: Serial Executions of Heterogeneous Computing of the Gauss-Seidel Iteration.

RTE の数値計算がこれらのステージに分割できることに着目し，ソフトウエア・パイプライン技法による並列処理により高速化を図る．すなわち，

- 第 $i+1$ ブロックをメモリから GPU に転送
- 第 i ブロックの行列乗算
- 第 $i-1$ ブロックを GPU からメモリに転送
- 第 $i-2$ ブロックを更新
- 第 $i-3$ ブロックを MPI で他プロセスと交換

は同時に実行してもよく，これらの処理が完了すれば i をひとつ増やし，同様に 5 つの処理を同時実行することができる (Fig. 3)．これら 5 つのステージはメモリおよび GPU の計算資源を共有することが望ましく，OpenMP での並列化が適する．さらに (S2) の `cublasDgemm()` が非同期 API であることから，これを起動した直後に別の CPU の処理を実行可能であり，4 つのスレッドで上記の 5 つを処理することができる．

3 つのステージ (S1)–(S3) はひとつづきの計算の流れをなすが，CUDA ではこれをストリームとよび，複数の ストリームを同時に実行する機構が提供される．同一の ストリームに属する処理は前の処理が完了してから次の処理が実行されるため，明示的な同期は不要である．また `cublasDgemm()` の実行と同時にメモリと GPU 間のデータ転送をおこなうには非同期転送の `cudaMemcpyAsync()` をもちいる．その際，メインメモリは `cudaHostAlloc()` で割当てられる Page-Locked

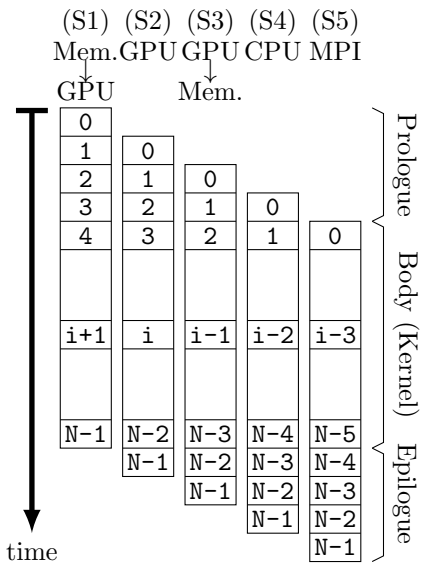


Fig. 3: Pipelining of Execusions.

メモリでなければならない．本研究では同一ブロックに対する (S1)–(S3) をひとつの CUDA ストリームとして合計 3 つのストリームを実行し，CPU に処理がうつる (S3) と (S4) の間でのみ同期すればよい．

5. 計算結果

本節では，以上の手順による計算事例を示す．計算環境には Table 2 に示す 4 ノードの PC をギガビット・イーサネットで接続してもちいる．

領域を $\Omega = (0,150) \times (0,150) \times (0,200)$ の矩形とし，領域全体で $\mu_s = 1.09$, $\mu_a = 0.08$ とした．散乱位相函数 p は Henyey-Greenstein 核 [1, 2]

$$p(\xi,\xi') = \frac{1}{4\pi}\frac{1-g^2}{(1-2g\,\xi\cdot\xi'+g^2)^{3/2}}, \quad g = 0.9$$

をもちいた．分割数を $N_1 = N_2 = 150, N_3 = 200, M = 1952$ とすると，領域内部の未知数は約 86 億個 (倍精度数で約 64 GB に相当) であった．また最大値ノルムでの残差が 10^{-13} 未満になるための Gauss-Seidel 法の反復回数は 320 回であった．ここで n^2 プロセスでの並列計算では，各プロセスで $\overline{\Omega}$ の部分集合 $[iL_1/n,(i+1)L_1/n]\times[jL_2/n,(j+1)L_2/n]\times[0,L_3]$, $i,j \in \{0,1,\ldots,n-1\}$ に対応する計算を割り当てた．

計算時間を Table 3 に示す．まず行列乗算に IMKL を利用し，全ての計算を CPU で実行すると 5.76 時間を要した．ここで並列計算は MPI のみをもちい，各 CPU の各コアごとに 1 つのプロセスを起動して合計 16 プロセスでおこなった．次に行列乗算に GPU を利用して Fig. 2 の手順のヘテロジニアス計算をおこなうと 2.27 時間を要した．ここでも並列計算には MPI のみをもちい，各ノードで 1 つのプロセスを起動して 4 プロセスでの並列計算をおこなった．さらにデータ転送時間等の隠蔽のため Fig. 3 に示す手順を 4 スレッド 4 プロセスのハイブリッド並列で実行すると 1.08 時間を要し，CPU のみの計算時間に対して 5.33 倍の高速化を達成した．

Table 3: Comparison of Computation Times of 3D RTE on $\Omega = (0,150)\times(0,150)\times(0,200)$ with $\mu_s \equiv 1.09$, $\mu_a \equiv 0.08$. The Number of Iterations of the Gauss-Seidel Method is 320.

Computing Devices	Time [Hours]	(Speed-up)
CPU (MPI)	5.76	(1)
CPU+GPU (MPI)	2.27	(2.53)
CPU+GPU (Hybrid)	1.08	(5.33)

演算ハードウエアの乗加算の実効処理速度を比較すると，CPU のみで 16 コアを利用する場合は 4.33×10^{11} ops/sec, これに 4 枚の GPU を加えると 2.86×10^{12} ops/sec となり，形式的な処理能力の向上は 6.61 倍である．そのもとで計算時間全体は 5.33 の高速化が達成されたが，これは乗加算の実効処理速度の向上の 81% に相当しており高いパフォーマンスを実現しているといえる．

また，先行研究 [12] のヒト頭部の MRI データを想定して $\Omega = (0,181)\times(0,217)\times(0,181)$ の矩形を考えて分割数を $N_1 = 181, N_2 = 217, N_3 = 181, M = 1952$ とすると，未知数は約 137 億個 (倍精度で約 102 GB) であった．Gauss-Seidel 法の反復を 3000 回としたときの計算時間を Table 4 に示す．この場合も CPU のみの計算時間に比して 5.83 倍の高速化を達成しており，提案方法の有効性がわかる．

Table 4: Comparison of Computation Times of 3D RTE on $\Omega = (0,181)\times(0,217)\times(0,181)$ with Biomedical Data. The Number of Iterations of the Gauss-Seidel Method is 3000.

Computing Devices	Time [Hours]	(Speed-up)
CPU (MPI)	80.4	(1)
CPU+GPU (MPI)	28.3	(2.84)
CPU+GPU (Hybrid)	13.8	(5.83)

6. ハイブリッド並列におけるメモリアクセスの帯域幅

提案するハイブリッド並列計算においては，Table 3, 4 に示すとおり計算全体では充分な高速化が得られている．一方で MPI のみのヘテロジニアス計算からのハイブリッド並列化だけに注目すると，前節のいずれの例においても 4 スレッドをもちいたにも関わらず，計算時間では約 2.1 倍の高速化にとどまっている．本節ではこの原因についてメモリアクセ

スに注目して論じる．

まず 5 節の前者の計算例において，Fig. 2 に示す計算を MPI のみによる並列計算において，Gauss-Seidel 法の 1 回の反復計算の各要素の計算時間とその割合を Table 5 に示す．同様に Fig. 3 をハイブリッド並列計算でおこなう場合のものを Table 6 に示す．また Fig. 4, Fig. 5 は，前節の 2 例に対し，各要素の計算時間を横軸にとって図示したものである．

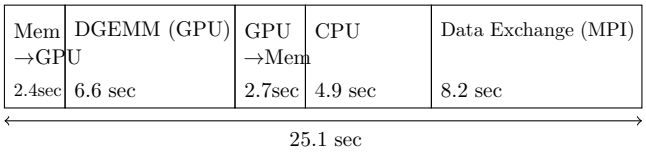
Table 5: Computational Time Details of Flat MPI Parallelization of the Procedure in Fig. 2 for $\Omega = (0, 150) \times (0, 150) \times (0, 200)$.

Operation	Time [sec]	
Memory→GPU	2.36	(9.4%)
DGEMM on GPU	6.61	(26.3%)
GPU→Memory	2.74	(10.9%)
CPU	4.91	(20.0%)
MPI data exchange	8.18	(32.6%)
Total	25.13	

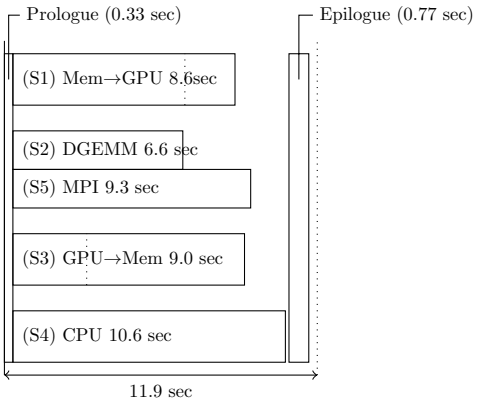
Table 6: Computational Time Details of Hybrid Parallelization of the Procedure in Fig. 3 for $\Omega = (0, 150) \times (0, 150) \times (0, 200)$.

Stage		Time [sec]
Prologue		0.33
(S1)	Memory→PL Memory	6.69
	PL Memory→GPU	1.95
(S2)	DGEMM on GPU	6.61
(S3)	GPU→PL Memory	2.87
	PL Memory →Memory	6.14
(S4)	CPU	10.60
(S5)	MPI data exchange	9.25
Epilogue		0.77
Total		11.86

まず Table 5 より，GPU での行列乗算ならびに MPI でのノード間データ通信がそれぞれ計算時間全体の約 3 割を占めていることから，Fig. 3 の並列化により理想的には 3 倍の高速化が期待される．一方 Table 6 より，ハイブリッド並列



(a) Flat MPI Execution Case (Fig. 2).



(b) Software Pipelining Execution Case (Fig. 3).

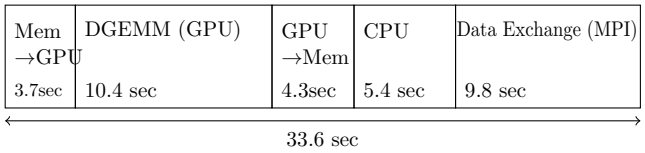
Fig. 4: Computational Time Details, $\Omega = (0, 150) \times (0, 150) \times (0, 200)$.

化によってメモリと GPU 間のデータ転送時間が，(S1) では 2.36 秒から 8.64 秒へ，(S3) では 2.74 秒から 9.01 秒へと 3 倍以上増加している．また CPU での計算 (S4) 時間も 4.91 秒から 10.60 秒へと 2.2 倍に増加している．また，生体データをもちい空間方向の分割数を大きくとる 5 節の第 2 例でも，この並列化による各ステージの計算時間変化に同様の傾向が見られた (Fig. 5)．

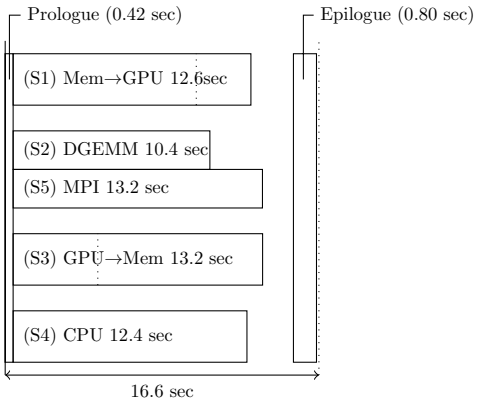
さらに詳細に検討するために，メモリアクセスの帯域幅に着目する．もちいた計算機のメモリ帯域幅を STREAM ベンチマーク [13] で測定した結果を Table 7 に示す．これより 4 スレッドでの実効帯域幅は約 15.6 GiB/sec であることがわかる．ここで 1 GiB = 1000^3Byte である．この計算事例では，各プロセスは約 22 億個の未知数を保持し，これは倍

Table 7: Effective Memory Bandwidth Measured by STREAM Benchmark, Where 1 GiB is 1000^3 Bytes.

Tests	Memory Bandwidth [GiB/sec] single thread	4 threads
Copy	13.9	14.7
Scale	14.1	14.7
Add	14.7	16.5
Triad	14.8	16.6



(a) Flat MPI Execution Case (Fig. 2).



(b) Software Pipelining Execution Case (Fig. 3).

Fig. 5: Computational Time Details, $\Omega = (0, 181) \times (0, 217) \times (0, 181)$.

精度で 17.6 GiB に相当する．(S1) ではメモリ間の転送および GPU への転送においてはこの 3 倍の量のメモリアクセスが発生するため，メモリへのアクセスの帯域は 6.1 GiB/sec となる．同様に (S3) は 5.8 GiB/sec であり，これらを合わせて 11.9 GiB/sec を占める．一方，CPU 演算部分 (S4) の帯域に関しては，Intel 社製のプロファイラではここでもちいた同社製の CPU におけるスレッド並列アプリケーションのメモリ帯域の飽和を検出しない [14] ことからその正確な計測は困難であると考えられる．しかしながら (S1), (S3) のみで実効帯域の大半を占めていることから (S1), (S3), および (S4) の速度低下はメモリアクセスの帯域幅の制約に起因すると考えられる．さらに以上の計算から，提案するハイブリッド並列では帯域幅を最大限まで使用する効率的な実装であることも示唆される．

7. 結言

本論文では，RTE の Gauss-Seidel 法による数値解法に対して，メモリアクセスと演算量に着目して，散乱積分の演算には GPU をもちい，差分などの演算には CPU をもちいるヘテロジニアス計算の高速化を提案した．ソフトウエアパイプラインによるデータ転送の隠蔽をスレッド並列により実装したところ，スレッド数の増大に対する速度向上比は充分ではなかったが，これはメモリ帯域幅の制限を受けるためであることを示した．しかし CPU および GPU の混在する環境において，乗加算の実効処理速度の形式的な増加が 6.61 倍であるところ，全体の計算時間では 5.33 倍から 5.83 倍の速度向上が得られ，効率よくハードウエアを利用していることが示された．

謝辞 本研究は科研費 (基盤研究 (B) No. 25287028, 基盤研究 (C) No. 26400198, 基盤研究 (C) No. 15K09920) の助成を受けました．

参考文献

(1) S. R. Arridge : Optical Tomography in Medical Imaging, *Inverse Problems*, **15** (1999), R41–R93.

(2) Y. Yamada and S. Okawa : Diffuse Optical Tomography: Present Status and its Future, *Optical Review* **21** (2014) pp. 185–205.

(3) L. Wang, S. L. Jacques and L. Zheng : MCML-Monte Carlo Modeling of Light Transport in Multi-Layered Tissues, *Computer Methods and Programs in Biomedicine* **47** (1995) pp. 131–146.

(4) Q. Fang and D. A. Boas : Monte Carlo Simulations of Photon Migration in 3D Turbid Media Accelerated by Graphics Processing Units, *Opt. Express* **17** (2009) pp. 20178–20190.

(5) Y. Fujii, T. Azumi, N. Nishio, S. Kato and M. Edahiro : Data Transfer Matters from GPU Computing, *2013 International Conference on Parallel and Distributed Systems*, pp. 275–282.

(6) Compute Unified Device Architecture, `http://docs.nvidia.com/cuda`

(7) A. D. Klose, U. Netz, J. Beuthan and A. H. Hielscher : Optical Tomography Using the Time-Independent Equation of Radiative Transfer — Part 1 : Forward Model, *J. Quantitative Spectroscopy & Radiative Transfer* **72** (2002) pp. 691–713.

(8) 藤原宏志 : 多重格子法による輸送方程式の定常問題に対する差分法の高速解法, 計算数理工学論文集 **11** (2011), pp. 13–18.

(9) Basic Linear Algebra Subprograms, `http://www.netlib.org/blas`

(10) IMKL : Intel Math Kernel Library, `https://software.intel.com/en-us/intel-mkl`

(11) The NVIDIA CUDA Basic Linear Algebra Subroutines library, `http://docs.nvidia.com/cuda/cublas`

(12) 藤原宏志 : 3 次元輻射輸送方程式の境界値問題の数値計算とその応用, 計算数理工学論文集 **12** (2012) pp. 43–48.

(13) J. D. McCalpin : STREAM : Sustainable Memory Bandwidth in High Performance Computers, `https://www.cs.virginia.edu/stream`

(14) Detecting Memory Bandwidth Saturation in Threaded Applications, `https://software.intel.com/en-us/articles/detecting-memory-bandwidth-saturation-in-threaded-applications`