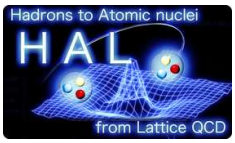
PROCEEDINGS OF SCIENCE

# An implementation of hybrid parallel CUDA code for the hyperonic nuclear forces

**Hidekatsu Nemura** *
*Center for Computational Sciences, University of Tsukuba, Tsukuba, Ibaraki, 305-8577, Japan*
*E-mail:* nemura.hidekatsu.gb@u.tsukuba.ac.jp

**for HAL QCD Collaboration**




We present our recent effort to develop a GPGPU program to calculate 52 channels of the Nambu-Bethe-Salpeter (NBS) wave functions in order to study the baryon interactions, from nucleon-nucleon to $\Xi-\Xi$, from lattice QCD. We adopt CUDA programming to perform the multi-GPU execution on a hybrid parallel programming with MPI and OpenMP. Effective baryon block algorithm is briefly outlined, which calculates efficaciously a large number of NBS wave functions at the same time, and three CUDA kernel programs are implemented to materialize the effective baryon block algorithm using GPUs on the single-program multiple-data (SPMD) programming model. In order to parallelize multiple GPUs, we take both two approaches by dividing the time dimension and by dividing the spatial dimensions. Performances are measured using HA-PACS supercomputer in University of Tsukuba, which includes NVIDIA M2090 and NVIDIA K20X GPUs. Strong scaling and weak scaling measured by using both M2090 and K20X GPUs are presented. We find distinct difference between the M2090 and the K20X in the sustained performance measurement of particular kernel executions which utilize the cudaStream objects.

*The 33rd International Symposium on Lattice Field Theory*
*14 -18 July 2015*
*Kobe International Conference Center, Kobe, Japan**

*Speaker.





PoS(LATTICE 2015)031

## 1. Introduction

Thanks to both elevating computer performance and various inventions of numerical algorithms, the lattice QCD approach to nuclear physics is being developed as a first-principle calculation. Not only two-body systems [1, 2, 3, 4] but also nuclear few-body systems [5, 6] are the playground for the present-day lattice QCD simulations. In addition, a new approach to the $NN$ interaction from the lattice QCD has been proposed[7, 8]. In this approach, the nucleon-nucleon ($NN$) potential can be obtained from the lattice QCD by measuring the Nambu-Bethe-Salpeter (NBS) wave function and the observables such as the phase shifts and the binding energies are calculated through the resultant potential[9]. This approach has been further extended and applied to various hadronic systems. See Ref. [10] and references therein for the state-of-the-art outcomes. Furthermore, a large scale lattice QCD calculation is now in progress [11] to study the baryon interactions from $NN$ to $\Xi\Xi$ by measuring the NBS wave functions for 52 channels.

The purpose of this paper is to present our recent effort to develop a hybrid parallel GPGPU program for multiple devices to perform the calculation of the baryon interactions. This report is organized as follows: Section 2 briefly outlines the effective block algorithm to calculate the NBS wave functions. Section 3 shows the machine and programming softwares used in this work. Section 4 is devoted to present the hybrid parallel program for multi-GPU calculation. In Sec. 5 we show the performances of the calculation with GPUs for the 52 channels of the NBS wave functions. Sec. 6 summarizes the report.

## 2. Formulation

In order to study the baryon interactions, the primary quantity we compute with lattice QCD is the four-point correlation function defined by

$$F^{\langle B_1B_2\overline{B_3B_4}\rangle}_{\alpha_1\alpha_2,\alpha_3\alpha_4}(\vec{r},t-t_0)=\sum_{\vec{X}}\left\langle 0\left|B_{1,\alpha_1}(\vec{X}+\vec{r},t)B_{2,\alpha_2}(\vec{X},t)\overline{\mathscr{J}_{B_{3,\alpha_3}B_{4,\alpha_4}}(t_0)}\right|0\right\rangle, \tag{2.1}$$

where the summation over $\vec{X}$ selects states with zero total momentum. The $B_{1,\alpha_1}(x)$ and $B_{2,\alpha_2}(y)$ denote the interpolating fields of the baryons such as

$$\begin{aligned}
&p=\varepsilon_{abc}\left(u_aC\gamma_5d_b\right)u_c, && n=-\varepsilon_{abc}\left(u_aC\gamma_5d_b\right)d_c, && \Lambda=\tfrac{1}{\sqrt{6}}\left(X_u+X_d-2X_s\right),\\
&\Sigma^+=-\varepsilon_{abc}\left(u_aC\gamma_5s_b\right)u_c, && \Sigma^0=\tfrac{1}{\sqrt{2}}\left(X_u-X_d\right), && \Sigma^-=-\varepsilon_{abc}\left(d_aC\gamma_5s_b\right)d_c,\\
&\Xi^0=\varepsilon_{abc}\left(u_aC\gamma_5s_b\right)s_c, && \Xi^-=-\varepsilon_{abc}\left(d_aC\gamma_5s_b\right)s_c,
\end{aligned} \tag{2.2}$$

where

$$X_u=\varepsilon_{abc}\left(d_aC\gamma_5s_b\right)u_c,\quad X_d=\varepsilon_{abc}\left(s_aC\gamma_5u_b\right)d_c,\quad X_s=\varepsilon_{abc}\left(u_aC\gamma_5d_b\right)s_c. \tag{2.3}$$

For simplicity, we have suppressed the explicit spinor indices and spatial coordinates in Eqs. (2.2) and (2.3); $\overline{\mathscr{J}_{B_{3,\alpha_3}B_{4,\alpha_4}}(t_0)}$ is a source operator which create $B_{3,\alpha_3}B_{4,\alpha_4}$ states at $t=t_0$. Hereafter, explicit time dependences are suppressed. In order to quantify the four-point correlation function $F^{\langle B_1B_2\overline{B_3B_4}\rangle}_{\alpha_1\alpha_2,\alpha_3\alpha_4}(\vec{r})$, we first consider the Wick's contraction together with defining the baryon blocks $[B^{(0)}_{1,\alpha_1}](\vec{x};\xi'_{P_1},\xi'_{P_2},\xi'_{P_3})$ and $[B^{(0)}_{2,\alpha_2}](\vec{y};\xi'_{P_4},\xi'_{P_5},\xi'_{P_6})$,

$$\begin{aligned}
F^{\langle B_1B_2\overline{B_3B_4}\rangle}_{\alpha_1\alpha_2,\alpha_3\alpha_4}(\vec{r}) \;=\; &\textstyle\sum_{\vec{X}}\sum_P\sigma_P\,[B^{(0)}_{1,\alpha_1}](\vec{X}+\vec{r};\;\xi'_{P_1},\xi'_{P_2},\xi'_{P_3})\,[B^{(0)}_{2,\alpha_2}](\vec{X};\;\xi'_{P_4},\xi'_{P_5},\xi'_{P_6})\\
&\times\varepsilon_{c'_1c'_2c'_3}\varepsilon_{c'_4c'_5c'_6}(C\gamma_5)_{\alpha'_1\alpha'_2}(C\gamma_5)_{\alpha'_4\alpha'_5}\delta_{\alpha'_3\alpha_3}\delta_{\alpha'_6\alpha_4},
\end{aligned} \tag{2.4}$$

with

$$\begin{aligned}[B^{(0)}_{1,\alpha_1}](\vec{x};\,\xi'_{P_1},\xi'_{P_2},\xi'_{P_3}) &= \left\langle B_{1,\alpha_1}(\vec{x})\;\overrightarrow{q}_{B_1,3}(\xi'_{P_3})\overrightarrow{q}_{B_1,2}(\xi'_{P_2})\overrightarrow{q}_{B_1,1}(\xi'_{P_1})\right\rangle, \quad \text{and}\\ [B^{(0)}_{2,\alpha_2}](\vec{y};\,\xi'_{P_4},\xi'_{P_5},\xi'_{P_6}) &= \left\langle B_{2,\alpha_2}(\vec{y})\;\overrightarrow{q}_{B_2,6}(\xi'_{P_6})\overrightarrow{q}_{B_2,5}(\xi'_{P_5})\overrightarrow{q}_{B_2,4}(\xi'_{P_4})\right\rangle,\end{aligned} \tag{2.5}$$

where $\sigma_P$ and $\{\xi'_{P_1},\cdots,\xi'_{P_6}\}$ are the sign factor and the set of permutated color-spin-space coordinates for each permutation $P$, respectively. Both 3-tuple sets of the quark fields $\{\overrightarrow{q}_{B_1,1},\overrightarrow{q}_{B_1,2},\overrightarrow{q}_{B_1,3}\}$ and $\{\overrightarrow{q}_{B_2,4},\overrightarrow{q}_{B_2,5},\overrightarrow{q}_{B_2,6}\}$ are ordered properly so as to correspond to the $B_1$ and $B_2$ states. Taking the expression in Eq. (2.4), the number of the iterations to obtain a $F^{\langle B_1B_2\overline{B_3B_4}\rangle}_{\alpha_1\alpha_2,\alpha_3\alpha_4}(\vec{r})$ reduces to $(N_c!N_\alpha)^B\times N_u!N_d!N_s!\times 2^{N_\Lambda+N_{\Sigma^0}-B}$, where $N_c=3, N_\alpha=4$ and $N_\Lambda$, $N_{\Sigma^0}$, $N_u, N_d, N_s$ and $B$ are the numbers of $\Lambda,\Sigma^0$, up-quark, down-quark, strange-quark and the baryons (i.e., always $B=2$ in the present study), respectively. In Ref. [12], only the limited spatial points were evaluated because of the computational cost $O(L^6)$ in the primitive numerical approach. After that we employed the Fast-Fourier-Transform (FFT) and the effective baryon blocks to improve the numerical performance to $O(L^3\log L^3)$ [13];

$$F^{\langle B_1B_2\overline{B_3B_4}\rangle}_{\alpha_1\alpha_2,\alpha_3\alpha_4}(\vec{r})=\sum_P\sigma_P\sum_{\vec{X}}\left([B^{(P)}_{1,\alpha_1}](\vec{X}+\vec{r})\times[B^{(P)}_{2,\alpha_2}](\vec{X})\right)_{\alpha_3\alpha_4}=\frac{1}{L^3}\sum_{\vec{q}}\left(\sum_P\sigma_P\left(\widetilde{[B^{(P)}_{1,\alpha_1}]}(\vec{q})\times\widetilde{[B^{(P)}_{2,\alpha_2}]}(-\vec{q})\right)_{\alpha_3\alpha_4}\right)\mathrm{e}^{i\vec{q}\cdot\vec{r}}. \tag{2.6}$$

See Ref. [14] how to extract the effective baryon blocks. For example, the specific form of the four-point correlation function $F^{\langle p\Lambda\overline{pX_u}\rangle}_{\alpha_1\alpha_2,\alpha_3\alpha_4}(\vec{r})$ of the $\langle p\Lambda\overline{pX_u}\rangle$ channel is given by,

$$\begin{aligned}F^{\langle p\Lambda\overline{pX_u}\rangle}_{\alpha_1\alpha_2,\alpha_3\alpha_4}(\vec{r})=\frac{1}{L^3}\sum_{\vec{q}}\Big(&[\widetilde{p}^{(1)}_{\alpha_1\alpha_3}](\vec{q})[\widetilde{\Lambda}^{(1)}_{\alpha_2\alpha_4}](-\vec{q})-[\widetilde{p}^{(2)}_{\alpha_1\alpha_4}]_{c'_3c'_6}(\vec{q})[\widetilde{\Lambda}^{(2)}_{\alpha_2\alpha_3}]_{c'_3c'_6}(-\vec{q})\\ &-[\widetilde{p}^{(3)}_{\alpha_1\alpha_3}]_{c'_2\alpha'_2c'_4\alpha'_4}(\vec{q})[\widetilde{\Lambda}^{(3)}_{\alpha_2\alpha_4}]_{c'_2\alpha'_2c'_4\alpha'_4}(-\vec{q})+[\widetilde{p}^{(4)}_{\alpha_1\alpha_4}]_{c'_1\alpha'_1c'_5\alpha'_5}(\vec{q})[\widetilde{\Lambda}^{(4)}_{\alpha_2\alpha_3}]_{c'_1\alpha'_1c'_5\alpha'_5}(-\vec{q})\\ &+[\widetilde{p}^{(5)}_{\alpha_1\alpha_3\alpha_4}]_{c'_1\alpha'_1c'_6}(\vec{q})[\widetilde{\Lambda}^{(5)}_{\alpha_2}]_{c'_1\alpha'_1c'_6}(-\vec{q})-[\widetilde{p}^{(6)}_{\alpha_1\alpha_3\alpha_4}]_{c'_3c'_5\alpha'_5}(\vec{q})[\widetilde{\Lambda}^{(6)}_{\alpha_2}]_{c'_3c'_5\alpha'_5}(-\vec{q})\Big)\,\mathrm{e}^{i\vec{q}\cdot\vec{r}}.\end{aligned} \tag{2.7}$$

By employing the effective block algorithm, the number of iterations to evaluate the r.h.s. of Eq. (2.7) except the momentum space degrees of freedom becomes $1+N_c^2+N_c^2N_\alpha^2+N_c^2N_\alpha^2+N_c^2N_\alpha+N_c^2N_\alpha=370$, which is significantly smaller than the number $(N_c!N_\alpha)^B\times N_u!N_d!N_s!\times 2^{N_\Lambda+N_{\Sigma^0}-B}=3456$ seen in Eq. (2.4). The manipulation on the expression of Eq. (2.6) in terms of the effective blocks $[B^{(P)}_{1,\alpha_1}]$ and $[B^{(P)}_{2,\alpha_2}]$ can be automatically done once the set of the interpolating fields (i.e., the quantum numbers) of both sink and source parts is given [14].

## 3. Machine and programming softwares

The present implementation is performed to utilize HA-PACS supercomputer in University of Tsukuba, which includes base cluster part and tightly coupled accelerators (TCA) part. The base cluster part consists of 268 nodes, each of which comprises two Intel E5-2670 CPUs as well as four NVIDIA M2090 GPUs connected by PCI-express, and started for common use in 2012. The TCA part involving 64 nodes was added to the HA-PACS in 2013, each of which comprises two Intel E5-2680v2 CPUs and four NVIDIA K20X GPUs. [1] Table 1 summarizes properties of these GPUs. For programming softwares on HA-PACS in this report we employed Intel C++ Compiler Version

[1] The TCA system is developed to implement a proprietary interconnect especially for accelerators, in order to shorten the communication latency among accelerators over different nodes[15].

| | Base cluster part | TCA part |
|---|---|---|
| Name | Tesla M2090 | Tesla K20Xm |
| Peak performance (GFlops, DP) | 665 | 1310 |
| Compute capability | 2.0 | 3.5 |
| Global memory (GiB) | 5.25 | 5.62 |
| ECC | Enabled | Enabled |
| Clock rate (GHz) | 1.30 | 0.732 |
| Memory bus width (bit) | 384 | 384 |
| Memory clock rate (GHz) | 1.85 | 2.60 |
| Constant memory (KiB) | 64 | 64 |
| Shared memory per block (KiB) | 48 | 48 |
| 32-bit registers available per block | 32768 | 65536 |
| Threads in warp | 32 | 32 |

**Table 1:** Several outputs from *cudaDeviceProp* and the peak performance values of double precision (DP) in GFlops obtained from HA-PACS.

14.0.4.211, Intel MPI Library 4.1 for Linux, and NVIDIA Cuda compiler driver version 6.5.14.

## 4. Implementation of hybrid parallel CUDA code for multiple GPUs system

In Ref. [14], we developed a hybrid parallel C++ program to calculate the 52 channels of the four-point correlation functions by using both MPI and OpenMP. The program works on either Bridge++ or CPS++, where both modified versions are employed. For Bridge++, feasibility of two frameworks, OpenCL and OpenACC, to utilize the GPU is discussed [16]. In this work, we adopt NVIDIA's CUDA programming for the first testbed implementation, because the target machine is HA-PACS comprising NVIDIA's Fermi and Kepler generation GPUs so that a better performance is expected with developing the CUDA programming than others. We also, in this work, aim to materialize the multi-GPU execution on the hybrid parallel programming with MPI and OpenMP by the single-program multiple-data (SPMD) programming model. In order to utilize multiple GPUs from a single program, we assign one MPI process to each GPU. Basic arithmetic of double-precision complex-numbers is implemented by hand with utilizing shared memory. Constant memory is employed to store runtime parameters and constant parameters. In the following, we describe three CUDA kernels implemented to calculate the 52 channels of the NBS wave functions.

**(i)** ***NormalBaryonBlocks*** : Assuming that the quark propagators are already solved, we first compute the normal baryon blocks on GPUs:

$$[B_\alpha^{(0)}](\vec{r};\ \xi_1',\xi_2',\xi_3') = \left\langle B_\alpha(\vec{r})\ \vec{q}_3'(\xi_3')\vec{q}_2'(\xi_2')\vec{q}_1'(\xi_1')\right\rangle, \quad \text{with} \quad B = p,\Sigma^+,\Xi^0,X_u,X_d,X_s, \qquad (4.1)$$

where three quark flavors $q_1', q_2', q_3'$ are appropriately chosen to create the corresponding $B$ state. The other baryon blocks, $B = n,\Sigma^-,\Xi^-,\Sigma^0,\Lambda$, are obtained from the above according to Eqs. (2.2) and (2.3) with presuming the symmetricity under the interchange of up and down quarks in the isospin symmetric limit. After the kernel execution, the FFT is employed to obtain the baryon blocks in momentum space. No clear benefit nor clear disadvantage is observed in performing the FFT whether on host side or on device side; the bottleneck is due to the *Alltoall* MPI communications for the FFT. The data of normal baryon blocks are replaced by its in momentum space after the FFT.

**(ii)** ***EffectiveBaryonBlocks*** : We construct the effective baryon blocks from the normal baryon blocks in momentum space,

$$\left\{ \widetilde{[B^{(d)}_{1,\alpha_1}]}_{\tilde{\boldsymbol{\xi}}_d}(\vec{q}), \widetilde{[B^{(d)}_{2,\alpha_2}]}_{\tilde{\boldsymbol{\xi}}_d}(-\vec{q}); \alpha_3, \alpha_4 \right\}, \qquad \text{with} \qquad B = p, \Sigma^+, \Xi^0, X_u, X_d, X_s, \tag{4.2}$$

where $\{\tilde{\boldsymbol{\xi}}_d\}$ denotes the indices which originate from the quark fields in the source; for example, in Eq. (2.7), it becomes $\{\text{none}\}$, $\{c'_3, c'_6\}$, $\{c'_2, \alpha'_2, c'_4, \alpha'_4\}$, $\{c'_1, \alpha'_1, c'_5, \alpha'_5\}$, $\{c'_1, \alpha'_1, c'_6\}$, $\{c'_3, c'_5, \alpha'_5\}$ for the six terms of the four-point correlator of the channel $\langle p\Lambda\overline{pX_u}\rangle$. In order to avoid the warp divergence, the diagramatical classification is performed throughout in the CPU code and the resultant data is aligned in Structure of Arrays (SoA) format which is transferred to the device prior to the kernel execution. This kernel execution has less timing performance impact though it is indispensable to connect the former part and the next part. Therefore we have not paid very much attention to improve the performance of this kernel.

**(iii)** ***MultiplicationEffectiveBlocks*** : Performed the kernel executions described in the above, we make the product of two effective baryon blocks on GPUs,

$$\left( \widetilde{[B^{(d)}_{1,\alpha_1}]}(\vec{q}) \times \widetilde{[B^{(d)}_{2,\alpha_2}]}(-\vec{q}) \right)_{\alpha_3\alpha_4} = \sum_{\tilde{\boldsymbol{\xi}}_d} \left( \widetilde{[B^{(d)}_{1,\alpha_1}]}_{\tilde{\boldsymbol{\xi}}_d}(\vec{q}) \, \widetilde{[B^{(d)}_{2,\alpha_2}]}_{\tilde{\boldsymbol{\xi}}_d}(-\vec{q}) \right)_{\alpha_3\alpha_4}. \tag{4.3}$$

For entire 52 channels of the NBS wave functions, we have to consider all of such summations tabulated in Tables 1-4 in Ref. [14]. This is one of the time consuming part of the present calculation. The symbolical manipulations are performed in the CPU code and the resultant SoA formed data is transferred to GPU to suppress the warp divergence prior to the kernel execution. To have good overlapping the kernel executions and the data transfer between the host and device, especially for the K20X GPU, we also utilize the *cudaStream* objects. The NBS wave function is finally obtained by performing the inverse FFT.

## 5. Results

Table 2 shows the performance values of double-precision computation in GFlops using single GPU for each single kernel execution of *NormalBaryonBlocks*, *EffectiveBaryonBlocks* or *MultiplicationEffectiveBlocks* with $L^3 \times T = 16^3 \times 32$ lattice measured on the base cluster part (M2090) or the TCA part (K20X). We also list the sustained GFlops values handled by *cudaStream* for the kernel executions of *NormalBaryonBlocks* and *MultiplicationEffectiveBlocks* in parentheses. The

| Kernel | Tesla M2090 | Tesla K20Xm |
|---|---|---|
| *NormalBaryonBlocks* | 98 (98) | 94 (95) |
| *EffectiveBaryonBlocks* | 3.1 | 0.55 |
| *MultiplicationEffectiveBlocks* | 8.2 (8.0) | 3.2 (27) |

**Table 2:** The performance values in GFlops using single GPU for each single kernel execution of *NormalBaryonBlocks*, *EffectiveBaryonBlocks* or *MultiplicationEffectiveBlocks*, measured on the base cluster part (M2090) or on the TCA part (K20X). The calculation is performed in an accuracy of double precision with the lattice size $L^3 \times T = 16^3 \times 32$. In parentheses the sustained performance values handled by cudaStream are shown for the kernel executions of *NormalBaryonBlocks* and *MultiplicationEffectiveBlocks* in GFlops.

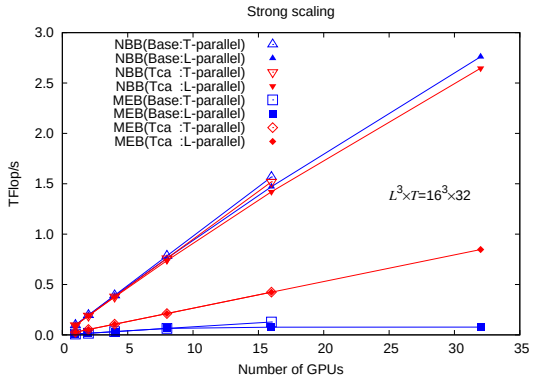



**Figure 1:** Strong scaling of the CUDA kernel executions with the lattice size $L^3 \times T = 16^3 \times 32$.

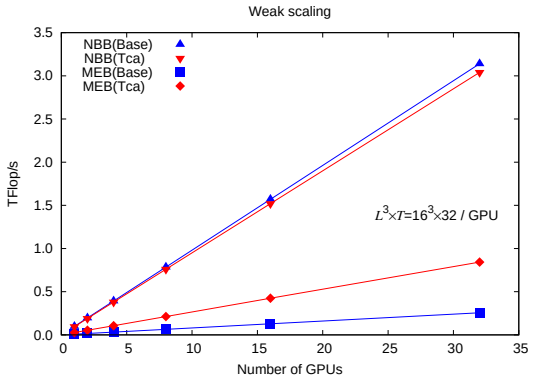



**Figure 2:** Weak scaling of the CUDA kernel executions with the lattice size $L^3 \times T = 16^3 \times 32$ per GPU.

handling of kernel executions for *MultiplicationEffectiveBlocks* by *cudaStream* lifts up the performance for the K20X more than factor 8 whereas no improvement is observed for the M2090. This is because the different architecture between the M2090 and the K20X; the compute capability of M2090 (K20X) is 2.0 (3.5). Figure 1 shows the strong scalings of two kernel executions *NormalBaryonBlocks* (NBB) and *MultiplicationEffectiveBlocks* (MEB) with total lattice size $L^3 \times T = 16^3 \times 32$ measured on the base cluster part (M2090) and the TCA part (K20X). In parallelizing across multiple GPUs, we take both two approaches by dividing the time dimension and by dividing the spatial dimensions, which are indicated by "T-parallel" and "L-parallel" in the figure. Detailed parameters which specifies the load on each GPU are adjusted on each measurement. Figure 2 shows the weak scalings of two kernel executions *NormalBaryonBlocks* (NBB) and *MultiplicationEffectiveBlocks* (MEB) for (an-)isotropic lattice with size $L^3 \times T = 16^3 \times 32$ per GPU measured on the base cluster part (M2090) and the TCA part (K20X).

## 6. Summary

In this paper, we present a recent effort to develop the hybrid parallel GPGPU program for multiple devices that calculates the 52 channels of the NBS wave functions. The implementation and the performance measurements are performed by using HA-PACS supercomputer in University of Tsukuba, which comprises the base cluster part including NVIDIA M2090 and the TCA part including NVIDIA K20X. In order to have better performance by using the GPUs, we adopt CUDA programming for the first testbed implementation. In performing the FFT, no clear benefit nor clear disadvantage is observed whether by using CPUs or by using GPUs. Three kernel programs are implemented by considering the data ordering on the device memory, suppression of the warp divergence, and making use of the shared memory and the constant memory. We also employ the cudaStream to perform efficiently the kernel executions as well as the data transfers between the host and device because the data transfers are indispensable for the large scale calculation of 52 channels of the NBS wave functions. The strong scaling and the weak scaling are measured for the kernel executions. Distinct difference between the M2090 and the K20X is observed in

the sustained performance measurement of the particular kernel executions; handling of kernel executions by using cudaStream is a key to make better use of latest GPUs in this approach.

## Acknowledgments

The author would like to thank CP-PACS/JLQCD collaborations and ILDG/JLDG [17] for allowing us to access the full QCD gauge configurations, and developers of Bridge++ [18], and the Computational Materials Science Initiative (CMSI). Calculations in this paper have been performed by using the HA-PACS computer under the Interdisciplinary Computational Science Program in CCS, University of Tsukuba. This research was supported in part by Strategic Program for Innovative Research (SPIRE), the MEXT Grant-in-Aid, Scientific Research on Innovative Areas (No. 25105505).

## References

[1] M. Fukugita, Y. Kuramashi, M. Okawa, H. Mino and A. Ukawa, Phys. Rev. D **52**, 3003 (1995).

[2] S. R. Beane, P. F. Bedaque, K. Orginos and M. J. Savage, Phys. Rev. Lett. **97**, 012001 (2006).

[3] S. Muroya, A. Nakamura and J. Nagata, Nucl. Phys. Proc. Suppl. **129**, 239 (2004).

[4] S. R. Beane *et al.* [NPLQCD Collab.], Nucl. Phys. A **794**, 62 (2007).

[5] T. Yamazaki *et al.* [PACS-CS Collaboration], Phys. Rev. D **81**, 111504 (2010).

[6] S. R. Beane *et al.* [NPLQCD Collaboration], Phys. Rev. D **87**, no. 3, 034506 (2013).

[7] N. Ishii, S. Aoki, T. Hatsuda, Phys. Rev. Lett. **99**, 022001 (2007).

[8] S. Aoki, T. Hatsuda and N. Ishii, Prog. Theor. Phys. **123** (2010) 89.

[9] S. Aoki *et al.* [HAL QCD Collaboration], PTEP **2012**, 01A105 (2012).

[10] K. Sasaki *et al.* [HAL QCD Collaboration], PTEP **2015**, no. 11, 113B01 (2015).

[11] T. Doi *et al.*, arXiv:1512.01610 [hep-lat]; N. Ishii *et al.*, in these proceedings; K. Sasaki *et al.*, in these proceedings;

[12] H.Nemura, N.Ishii, S.Aoki and T.Hatsuda, Phys. Lett. B **673**, 136 (2009).

[13] H. Nemura, N. Ishii, S. Aoki and T. Hatsuda [PACS-CS Collaboration], PoS **LATTICE2008**, 156 (2008).

[14] H. Nemura, arXiv:1510.00903 [hep-lat].

[15] T. Hanawa, Y. Kodama, T. Boku and M. Sato, IPDPSW '13 Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, 1030 (2013).

[16] S. Motoki, *et al.*, in these proceedings.

[17] See `http://www.lqcd.org/ildg` and `http://www.jldg.org`

[18] Lattice QCD code Bridge++, `http://bridge.kek.jp/Lattice-code/index_e.html`.