mSMS: PGAS Runtime with Efficient Thread-based Communication for Global-view Programming

Hiroko Midorikawa, Kenji Kitagawa, Yugo Sakaguchi
Department of Computer and Information Science, Seikei University
Tokyo, Japan
midori@st.seikei.ac.jp, kitagake@ci.seikei.ac.jp, dm186204@cc.seikei.ac.jp

***Abstract*— A new partitioned global address space (PGAS) runtime, mSMS, with efficient thread-based communication is proposed in this work. The mSMS runtime system transparently invokes system threads behind a user program, to support the concurrent execution of remote communication and calculations, and manage dynamic thread creation/deletion in the user program. Two types of applications—stencil computations with synchronous static data access and Barnes–Hut problems with asynchronous dynamic data access—are evaluated. The mSMS achieves performance comparable to or better than that of MPI and other PGAS languages.**

Keywords—PGAS, global address space, software-distributed shared memory, global-view programming, cluster, runtime system

I. Introduction

Many partitioned global address space languages (PGASs) do not provide a genuine global address space (GAS); rather, they provide a global name space for the data shared among the computing nodes. In some PGASs, the accessible area in the shared data space is limited to the boundary sleeve area between nodes and/or pointer-based access is not available [1]. In UPC [2], pointer-based access to global data is available; however, the global pointer to global data has a different datatype from that of the local pointer to local data. Most of the PGASs convert such a different data-access notation to an explicit communication to a remote node, e.g., one-sided MPI, by static analysis in its dedicated compiler. This strategy is effective for applications with static and synchronous access to predefined data areas, such as stencil computations. However, it is not sufficient for applications with temporally and spatially dynamic data access. In such cases, runtime mechanisms that support concurrent computing and communication among user threads are necessary. One way to resolve this is by introducing application-specific control codes into each application [3]; however, this results in low productivity during PGAS program development, as in the case of MPI program development.

II. mSMS with Efficient Thread-based Communication

mSMS provides APIs and SMS library functions for an efficient runtime system that realizes a GAS for a large number of nodes in a cluster system, as shown in Fig. 1. Each thread of all SMS processes in nodes can access any part of the GAS using a consistent address in an ordinary C pointer. Certain areas of the GAS are allocated in the local node memory as *owner-pages*, and the other areas in the GAS are allocated in remote node memories. When a thread accesses the remote areas, remote SMS pages are cached in a local node as *cached pages.* When the memory-consistency synchronization function is called, the modified parts of the cached pages are transferred to the original owner node and the cached pages are discarded. The available size of the GAS is defined by the product of the available local node memory capacity and the number of nodes in use.

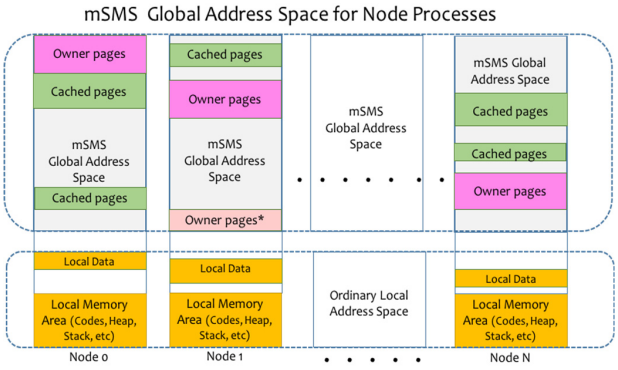

Fig. 1. mSMS global address space

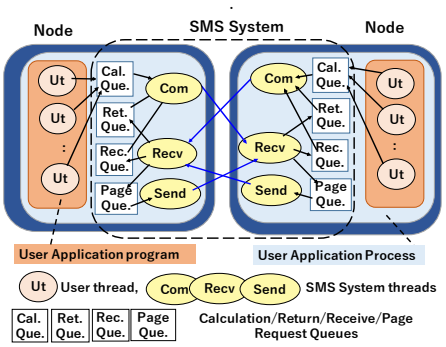

Fig. 2. mSMS system overview

Fig. 2 shows the internal of the SMS process, which includes three SMS system threads: 1) *comthread* that manages the internal requests from local user threads and external requests from remote nodes, 2) *receive-thread* that receives asynchronous messages from remote nodes, and 3) *send-thread* that sends the requested SMS pages to remote nodes. These system threads are invoked transparently when the sms_startup() function is called. They work as an efficient runtime system to support the concurrent execution of remote asynchronous communications and calculations by user threads, maintain data consistency among threads, and manage dynamic thread creation and deletion by a user program. This SMS

This work is funded by JHPCN-HPCI (Prject ID: jh190039-ISH) and JSPS KAKENHI (Grant Number JP18K11327)

978-1-7281-4734-5/19/$31.00 ©2019 IEEE

transparent runtime alleviates the burden on programmers, so that they can concentrate on the algorithms of their applications without worrying about the control of concurrent communications and calculations among the many threads in the many nodes. Fig. 3 shows an SMS program for 3D 7-point stencil computation using SMS shared data statements and SMS library functions. It is translated to a general C program by a simple SMS translator. The SMS is available without a dedicated compiler, unlike the ordinary PGASs.

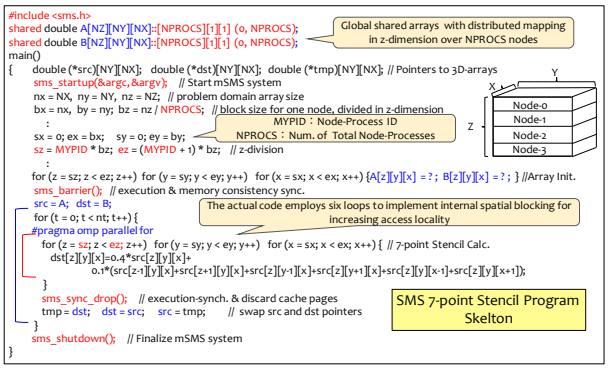

Fig. 3. SMS program for 7-point 3D-stencil computations.

III. PERFORMANCE EVALUATION

The performances of two different types of applications are shown here. One is a stencil computation with synchronous access to regular structured data, and the other is an N-body problem using the Barnes–Hut [5] with asynchronous access to irregular structured data. A Tsubame3.0 [6] super computer is used for this evaluation.

Fig. 4 shows the strong scaling performances of one step time of the 2D 5-point stencil computation in UPC [2], XcalableMP [1], MPI, and mSMS. The data size employed here is relatively small, 64 GB, because of the internal bit length limitation of the UPC global pointer. Each language uses its optimal number of threads in one node, from 8 to 32 threads [4]. The performances of UPC using the global pointer and local pointer are the worst and second worst, respectively. The mSMS achieves the best performance; which is almost the same as that of MPI. The XcalableMP also achieves comparable performance to that of mSMS, except in the case using two nodes. Fig. 5 shows the weak scaling performance for 3D 7-point stencil computation for large data, in the size of 0.26 TB – 23.0 TB, using 2 to 180 nodes in MPI and mSMS. The mSMS achieves better or comparable performances, when compared to MPI. An mSMS process updating a 128 GB data array in local memory in each node shares a 30 TB GAS among 180 nodes. Fig. 6 shows the performance of an n-body problem (1M – 512M bodies) in 3D space using mSMS with 64 nodes (processes) × 32 threads. This is a straightforward implementation of the Barnes–Hut [5] algorithm, which uses shared tree data allocated in mSMS GAS. Each thread in the nodes accesses the tree asynchronously, using an ordinary C pointer for its force calculation. Ordinary MPI implementation employs the Local Essential Tree (LET) method [5], where each MPI process first identifies the necessary LET data for its force calculation and collects the data from remote nodes before the calculation. On the other hand, the mSMS

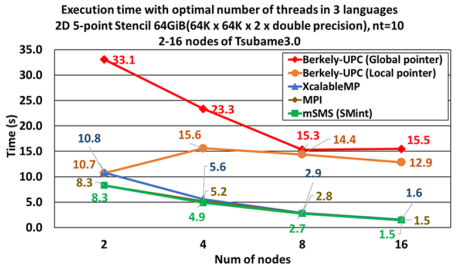

Fig. 4. Strong scaling performances of 2D 5-point stencil computations. One step time for 64GB data in UPC, XcalableMP, MPI, and mSMS

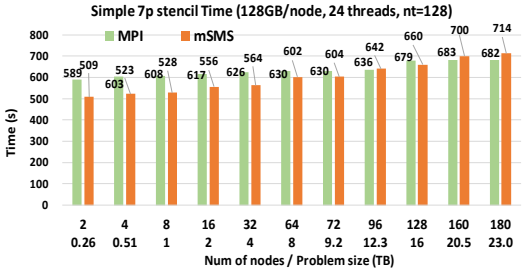

Fig. 5. Weak scaling performances of 3D 7-point stencil computation. mSMS+OpenMP vs. MPI+OpenMP
(One step time for 256GB – 23TB data using 2 – 180 nodes)

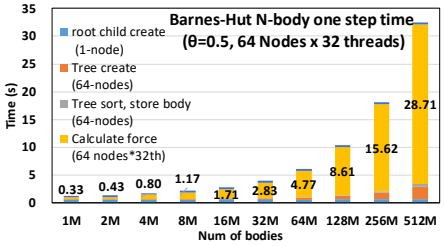

Fig. 6. Barnes–Hut n-body problem one step time (mSMS).

threads directly access the tree while the force calculation. The mSMS program is very simple, when compared to the UPC implementation [3] where complicated runtime control codes are introduced in the user program. In the mSMS, the shared tree is distributedly mapped in each node based on the space filling curve (SFC) to increase the access locality in each node. The performance of the current naïve version of the mSMS program is comparable to that of the UPC implementation.

REFERENCES

[1] XcalableMP https://xcalablemp.org/ [online 8/23/2019]

[2] Berkeley UPC ver.2.28.9 http://upc.lbl.gov/ [online 8/23/2019]

[3] J. Zhang, B. Behzad, and M. Snir, "Design of a multithreaded Barnes-Hut algorithm for multicore clusters," IEEE Trans. Parallel Distrib. Syst., vol. 26 , no. 7, pp.1861–1873, 2015.

[4] Y. Sakaguchi and H. Midorikawa, "The programmability and performance of global-view programming API: SMint for multi-node and multi-core processing," IEEE Pacific Rim Conf. Commun. Comp. Signal Proc., 2019.8

[5] J. K. Salmon, "Parallel implementation of the BH algorithm," PhD. dissertation, Phys., Math. Astron. Dept., California Inst. Technol., Pasadena, CA, USA, (1991).

[6] Tsubame3 http://www.gsic.titech.ac.jp/en, [online 8/23/2019]