Invasive MPI on Intel's Single-Chip Cloud Computer*

Isaías A. Comprés Ureña[1], Michael Riepen[2], Michael Konow[2], and Michael Gerndt[1]

[1] Technical University of Munich (TUM), Institute of Informatics, Boltzmannstr. 3, 85748 Garching, Germany
{compresu,gerndt}@in.tum.de
[2] German Microprocessor Lab, Intel Labs Braunschweig, Theodor-Heuss-Str. 7, 38122 Braunschweig, Germany
{michael.riepen,michael.konow}@intel.com

Abstract. The Single-chip Cloud Computer (SCC) from Intel Labs is an experimental CPU that integrates 48 cores. As its name suggests, it is a distributed memory system on a chip. In typical configurations, the available memory is divided equally across the cores. Message passing is supported by means of an on-die Message Passing Buffer (MPB). The memory organization and hardware features of the SCC make it an interesting platform for evaluating parallel programming models. In this work, an MPI implementation is optimized and extended to support the invasive programming model; the invasive model's main idea is to allow for resource aware programming. The result is a library that provides resource awareness through extensions to MPI, while keeping its features and compatibility.

Keywords: programming models, MPI, dynamic process management.

1 Introduction

A large percentage of the area of a CPU is used today for supporting memory coherence. There are commercial and experimental many-core chips that explore the option of doing away with coherency and increasing the number of computing elements. One of these experimental chips is the one used for this work: the Single-chip Cloud Computer (SCC) from Intel Labs.

The SCC[1] is a 48-core 'concept vehicle' created by Intel Labs as a platform for many-core software research. As its name suggests, the SCC is a distributed memory system on a chip; typically, memory on the SCC is divided equally and configured to be private to each core. In addition, each core has a Message Passing Buffer(MPB), which consists of 8KBs of directly addressable SRAM; these on die buffers can be used for low latency and high bandwidth communication, since they are visible by all cores. The cores are based on the P54C design and

* Support for this work was provided by the Transregional Collaborative Research Centre 89: Invasive Computing (InvasIC)[12].

A. Herkersdorf, K. Römer, and U. Brinkschulte (Eds.): ARCS 2012, LNCS 7179, pp. 74–85, 2012.

have moderate performance at 800MHz. Given the capability of these cores, results of parallel applications on the SCC are better evaluated in terms of scaling rather than absolute performance.

The scaling of a parallel application depends strongly on its available parallelism. This parallelism typically changes during different stages of the execution and can be input dependent; furthermore, adaptive refinement techniques can create load imbalances that only become apparent at runtime. In many cases, the parallelism that will be available in an application (given its parameters and input files) is not known before job launch; in spite of that, MPI applications today are started and finished with a static number of processes and lose efficiency as a consequence.

Dynamic creation of processes is available since version 2 of the MPI standard, yet this feature is rarely used in large clusters due to the latencies involved when creating new processes. Some MPI implementations do not support dynamic processes at all; current batch schedulers operate in ways that make it difficult to add dynamic processes support. A result of the lack of these features is that many applications run with a fixed number of processes and use load balancing techniques that do not rely on spawn operations. An MPI library and process manager combination that can handle processes dynamically and efficiently would allow for different load balancing strategies and more efficient use of resources. It is also beneficial to have a simple API that targets the manipulation of processes.

Within the Transregional Research Centre InvasIC[12], researchers of the Karlsruhe Institute of Technology, the University of Erlangen and the Technical University of Munich, are working on invasive programming. The goal is to enable resource aware programs to allocate and free hardware resources during execution. Invade, infect and retreat operations can be used to allocate, execute on, and free resources[7]. Research efforts investigate the approach at different levels of abstraction: from hardware support up to programming models. Invasive MPI (iMPI) is a first step towards invasive programming with MPI. This library extends the MPI-2 interface with the three invasive operations and includes a new process manager to reduce their latency. It is based on RCKMPI[4] and its SCCMPB channel; the channel is modified to improve scaling, support dynamic processes and share the MPB with the process manager. Compatibility with MPI-2 is kept and improvements also benefit standard applications.

2 Related Work

The SCC's distributed memory architecture maps naturally to message passing approaches. There are already MPI standard as well as non standard libraries available, for message passing research on the SCC.

2.1 RCCE

RCCE[2] is a minimal overhead library that provides message passing functionality on the SCC. It also provides low level access to the SCC's hardware many

features. RCCE exposes these features through a simple and clean interface to the programmer.

The message passing functions, while not being standard based, are similar to the blocking variants found on MPI. There are several community supplied extensions to RCCE. There is a non-blocking extension called iRCCE[3], developed at RTWH Aachen, that is used for the MPI library called SCC-MPICH[8].

In addition to the message passing functionality, RCCE also provides for: power management, configuration register access, shared memory, hardware locks, and other features. RCCE exposes the complete set of functionalities of the SCC to the programmer.

2.2 RCKMPI

Since the introduction of the RCKMB driver for the SCC, MPI has been possible on the chip through MPI libraries configured to use TCP/IP sockets. Both MPICH2 and Open MPI have been confirmed to be working reliably this way; however, their performance has been shown to be lacking when compared to RCCE and derivatives, in both latency and sustained bandwidth[5]. Because of the large number of parallel applications and developer tools, a well performing MPI library was identified as a desirable addition to the software offered with the SCC. The Rock Creek MPI (RCKMPI) library provides exactly that, performance that is comparable to RCCE in both latency and bandwidth, and compatibility with projects that use MPI directly or depend on libraries that require it.

RCKMPI[4] is based on MPICH2, and preserves its layered architecture. Its main feature is the addition of three SCC specific channels. The project uses both possible buffer types available on the SCC for core to core communication: the on die Message Passing Buffer and off chip shared memory.

3 The Invasive MPI Library

The Invasive MPI library aims to be an MPI-2.2 compliant library while extending the API with non-standard invasive operations. The use of invasive operations results in the reservation, creation or destruction of processes; therefore, they require low latency process management if they are to be used frequently in a parallel application. In this section, the internal characteristics of the Invasive MPI library are presented. Its general architecture and its custom process manager are described. The invasive extensions are later introduced and their interaction with the process manager is explained.

3.1 General Architecture

The library is based on the 1.4.0 release of MPICH2[11]. The layered architecture of MPICH2 is conserved; however, both internal and external interfaces are modified to accommodate for the invasive extensions. A modified SCCMPB channel

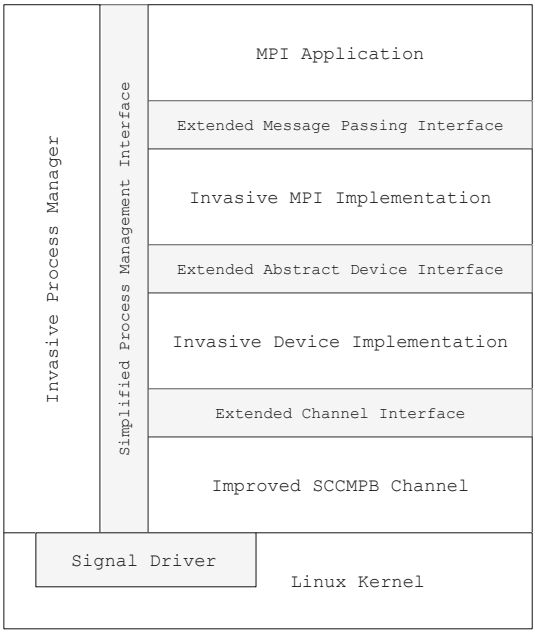


Fig. 1. Invasive MPI Architecture

from RCKMPI is used. The new channel allows for the MPB to be shared with the process manager; this eliminates the use of TCP/IP and Unix sockets entirely. The library runs on SCC Linux; MPI performance has been shown to be comparable to bare-metal for MPI jobs, for up to 4KB messages (as presented in [10], where RCKMPI on SCC Linux is compared to ETI's bare-metal framework).

There are three horizontal layers in the MPICH2 architecture (as shown in figure 1): the MPI, device and channel layers. The MPI layer is the one that interfaces with the program. In the device layer is where most of the MPI functionality is implemented (for example: point-to-point routines, collective operations, etc.). The channel layer implements non-blocking point-to-point communication; this functionality is used by the device layer to implement the operations described in the MPI standard.

Process management is arranged vertically in the architecture; operations can be done in all layers through the Process Management Interface (PMI). From MPICH2, different implementations of the PMI are available. Three are currently included with MPICH2: SIMPLE, SLURM, and SMPD. The different PMI implementations allow the library to work with specific process managers.

3.2 Invasive Process Manager

One of the main goals of the Invasive MPI library is to provide low latency process management. Low latency is required by the invasive operations for their

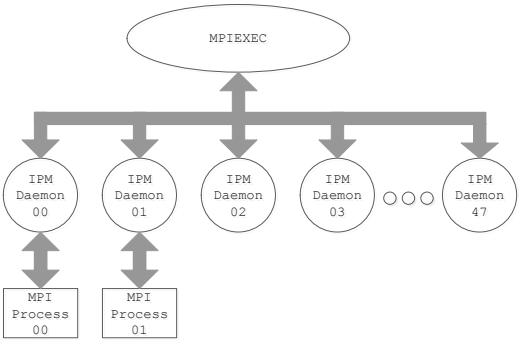


Fig. 2. Process manager state after launching two processes

use to be feasible in real applications. A process manager can be considered a separate project in itself; therefore, before making implementing one a goal, the available MPICH2 process managers were surveyed.

There are several process managers available that are compatible with the MPICH2 library. Some of them are part of the library itself, like: MPD, Hydra, GForker and SMPD. Others are external projects, such as: SLURM and batch schedulers. Some of these process managers did not work at all on the SCC, like in the case of the GForker, in terms of launching parallel applications across cores. The vast majority are able to work through the use of TCP/IP sockets, but with launch latencies that were deemed unacceptable for invasive operations. The SMPD, Hydra and MPD process manager were tested; the MPD provided the best compatibility and performance on the SCC (used for comparison in section 4: Process Management Performance).

A new process manager was identified as a necessary addition for the Invasive MPI library. Such a process manager should allow for low latency when launching MPI jobs, as well as when doing spawn operations. It must also provide functionality for resource awareness, topology information and power management. The Invasive Process Manager (IPM) aims to achieve that. This process manager is implemented as three components: the MPIEXEC program, the IPM daemons and a Linux kernel driver (built as a module). Figure 2 shows the IPM state of a 2 process MPI job.

To initialize the process manager, the kernel module must be loaded and the daemon must be started at each of the 48 cores of the SCC system. The module's task is to send POSIX Real-Time signals to processes running in remote cores (on the SCC, a similar approach was introduced earlier by the Distributed S-Net project[6]), while the daemon is responsible for responding to these signals appropriately.

Inter-core communication between the MPIEXEC program, the IPM daemons and MPI processes is done through the use of the MPB for payload and the Linux driver for notification. The driver converts a hardware interrupt into a signal and delivers it to the specified process. A user space library is used to

control the driver. The library provides asynchronous and synchronous modes of operation. The asynchronous mode consists of two calls: *sigmod_command* and *sigmod_result*. A typical use case for this mode of operation is to trigger several cores in parallel, and then collect their responses. The third call is *sigmod_command_sync*. Its implementation consists of the asynchronous calls used together with the atomic register (a hardware feature of the SCC) of the target core. The use of the MPB and the driver eliminates the use of TCP/IP or Unix sockets entirely, and results in a significant reduction of the latency of process management operations.

Launching MPI jobs on the SCC running SCC Linux is similar to spawning processes on on a cluster, in spite of them being cores in the same silicon. When launching a job, the MPIEXEC program signals the daemons to fork the processes, effectively creating a process group (PG). MPIEXEC's job is to survey the daemons and make decisions accordingly. The main purpose of an IPM daemon, running at a core in the SCC, is to fork and monitor a single MPI process in response to MPIEXEC's commands. Limiting the daemon to one process is a sensible design decision when taking into account the limited performance provided by a P54C core and the averse effects of locking when sharing resources (like the MPB). In the event of an MPI process failure, the IPM daemon receives a SIGCHLD and notifies MPIEXEC, which in turn terminates the parallel job by notifying all other participating daemons.

Process management operations are not always initialized by MPIEXEC; *MPI_Comm_spawn* (or invade and infect operations in the invasive model) operations are made directly by an MPI program. In this case, the PMI is used by the MPI processes; they communicate with the MPIEXEC program through their local IPM daemon, and ask for the extra process groups. In the case of the invasive model, the MPI application may also inquire on available resources, reserve them and later release them. PMI calls are also done in *MPI_Init* and *MPI_Finalize* (and retreat operations in the invasive case).

3.3 Invasive Extensions to MPI

The main objective of the Invasive MPI library is to support the invasive programming model with message passing. All features introduced previously are related to performance, scaling or flexible use of resources; the aim is to efficiently execute tasks required by the invade, infect and retreat operations.

The invasive operations are related to the creation and destruction of processes. They also modify the local process group and the world communicator. The three elemental operations were added to the MPI layer as: *MPI_Comm_invade*, *MPI_Comm_infect* and *MPI_Comm_retreat*.

The main purpose of invade is to reserve resources. For this, a survey to find out what is available must be done. In the current state of the Invasive MPI library for the SCC, the cores and their private memory are invaded as a single resource.

Typically, invade is followed immediately by an infect call. With infect, an application specifies the total number of cores to infect and which ones are

preferred; this number can be equal or less to the total cores that were reserved by the invade call. The infect algorithm is similar to that of *MPI_Comm_spawn*. A notable difference in the interface is the absence of the communicator and intercommunicator parameters. There are two reasons for this: first, no new intercommunicator is created as a result of infect; and second, the operation is always done across the world communicator. After an infect, processes can communicate with each other and do collective operations through the world communicator, as if they were always part of the originally launched process group by MPIEXEC.

The last operation is retreat. The retreat operation does the opposite of the invade plus infect sequence. Instead of reserving and claiming resources, it returns them so that other invasive applications can claim them.

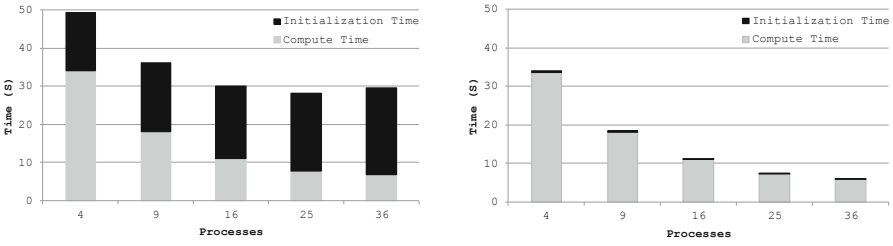
4 Process Management Performance

Most of the development done for the Invasive MPI library is connected to process management. One of the main goals was to lower the time required for the creation of processes. As a result, the latency of launching MPI jobs was reduced. To measure this effect, the NAS BT benchmark for size W is presented; compute time is shown together with initialization time. The initialization time includes process creation time and the *MPI_Init* call (which depend on the process manager), as well as the benchmark setup.

For comparison with existing process managers, results for the RCKMPI library are included. RCKMPI was configured with the well known Multiple Purpose Daemon (MPD) from MPICH2. The MPD uses SSH over TCP/IP when launching processes, while the IPM uses the MPB directly together with the new Linux driver. For PMI communication between a child MPI process and the local daemon, each MPD daemon sets up a local socket while the IPM uses named pipes and signals.

The Rocky Lake SCC systems were configured with 1600MHz for the routers and mesh, 800MHz for the cores and 1066MHz for the memory. SCC Linux 2.6.38, the driver, all libraries and benchmarks were compiled with GCC and Gfortran version 4.5.

In figure 3 the total time as seen by the user is shown for both the RCKMPI and Invasive MPI libraries. For this benchmark, it can be observed that the overall run time for both libraries is quite different after taking into account the process creation and initialization times. It is interesting to see that for the 36 process case (figure 3(a)), RCKMPI is actually slower than for the 25 process case, due to the increased overheads of the process manager when launching the extra 11 processes; in the case of the Invasive MPI library (as shown in figure 3(b)), there is a net gain when going from 25 to 36 processes. These results show that the reduction in MPI job launch times by the IPM is significant when compared to existing process managers (considering the MPD, Hydra and SMPD perform similarly on the SCC).



(a) RCKMPI (b) Invasive MPI

Fig. 3. Initialization and compute time for NAS 3.3 BT size W

5 Invasive Operations Performance

The development done for the Invasive MPI library was targeted at reducing the latency of operations required by the invade, infect and retreat routines of the invasive programming model. In this section, their final performance is evaluated. The results shown in this section were obtained with the same settings described in the process manager tests (section 4).

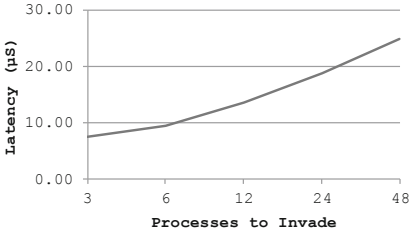
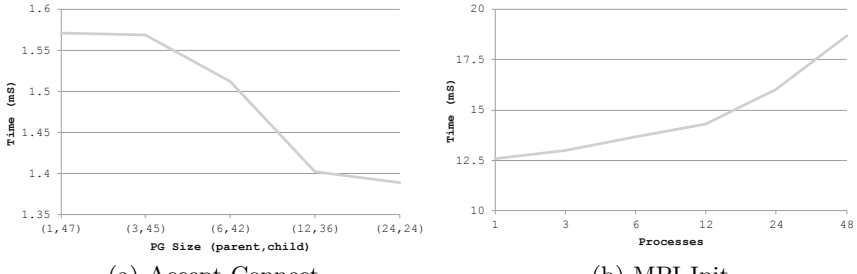


Fig. 4. Invade scaling

Invade involves the survey and reservation of resources. As previously mentioned, currently only the cores (together with their private memory) of the SCC system are surveyed and reserved. As shown in figure 4, the invade operation is under 30 microseconds for the worst case; the invade operation can be used frequently by an application before deciding to make an infect.

The infect operation involves a series of routine calls. The parent processes do what is essentially a spawn operation; this spawn operation differs from the standard one in that it modifies the world communicator and local process group,



(a) Accept-Connect (b) MPI_Init

Fig. 5. Scaling of operations required by infect

so that it contains the aggregated parent and child processes. The parent process group must wait for the *fork*, *exec* and *MPI_Init* routines to complete on the child process group, in addition to the process manager overhead, before proceeding; the result is an operation that is collective across both process groups.

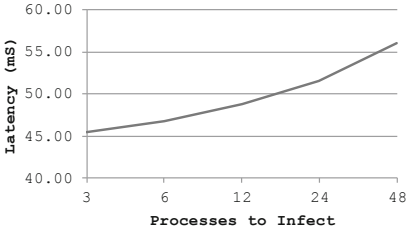


Fig. 6. Infect scaling

After the required processes are created, the parent process group (PG) makes an accept operation collectively; this call is matched with a connect operation at the children PG. Custom versions of the connect and accept operations were implemented to reduce their latency on the SCC. Figure 5(a) shows the latency of these operations for combinations of parent and child PG sizes that add up to 48 processes. The latencies for these operations are symmetric so only the first combinations are shown.

Another related operation to be measured is *MPI_Init*. Initialization scaling with the number of processes is shown in figure 5(b). Since for 1 process no communication is involved, 12.5 milliseconds is the minimal compute time required for this operation; the *fork* and *exec* operations also involve no communication, and together with *MPI_Init* their times add up to the total of approximately 46 milliseconds required to spawn a single process.

Scaling of the infect operation itself is shown in figure 6. The difference from infecting the minimal amount to the maximum is around 11 milliseconds; most of the increment in time is related to the additional synchronization required to initialize the process group and communicators. The *fork* and *exec* operations are started asynchronously so that most of their execution time overlap.

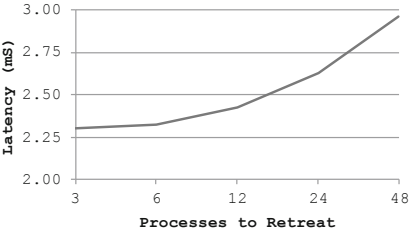


Fig. 7. Retreat scaling

The retreat operation is significantly faster than infect, but not as fast as invade. The operation requires a relatively small amount of communication. The retreat operation results in a retreating process group and a staying one. In most cases, the important value to measure is the time observed at the process group that stays. When doing an infect immediately after a retreat, the time required by the retreating PG is also important; under such scenarios, the invade operation will not be able to reserve the cores, until the processes terminate completely. Figure 7 shows scaling of the retreat operation when measured at the staying process group. The retreat operation is faster than infect by more than a factor of 10, in the worst case.

6 Invasive Numerical Integration

Numerical integration is a technique used to determine definite integrals. As with other problems in mathematics, an analytical solution for the integral of a function may be impossible and the numerical approach the only alternative. In this section, a parallel numerical integration algorithm is presented. This algorithm is a combination of the adaptive Simpson's method[9] and the midpoint rule. The hardware, software and configuration for the tests in this section are again the same as for the process manager tests (section 4).

The adaptive Simpson's method is typically implemented serially by means of recursion. The algorithm generates an unbalanced binary tree based on the input function. This tree is a result of refinement on intervals were the error is estimated to be large; the refinement is done by means of a bisection of integration subintervals. In order to estimate the error, the Simpson estimate for three and six points are combined.

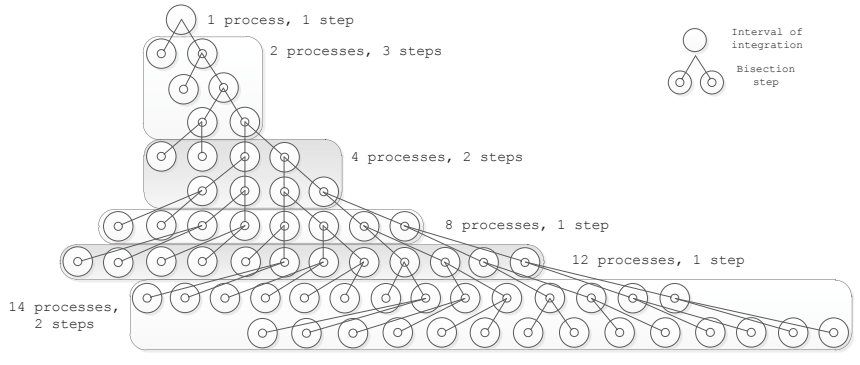


Fig. 8. Generated unbalanced tree and process count

When applying the midpoint rule, an integral is approximated by evaluating the value of the function at the middle of a subinterval; this value is then multiplied by the width of the subinterval to generate a rectangle. With this method, a large number of points need to be evaluated to obtain a decent estimate of the definite integral (depending of the function).

The algorithm presented in this section utilizes the bisection strategy of the adaptive Simpson's method. Once bisections are done and the bisection criterion indicates that the subinterval can be evaluated with an acceptable margin of error, the midpoint rule is used with 1,000,000 points. The bisection strategy dictates the number of processes that can be used at any given time. To adjust the number of processes, the invasive extensions of the library were used.

Figure 8 shows the particular binary tree generated by the function:

$$f(x) = \begin{cases} 1 & : x \leq 0 \\ e^x & : x > 0 \end{cases}$$

for the interval $(-100, 15)$. Each rectangle represent parts of the runtime where the parallelism is the same. Running the serial version of the algorithm took 11.932 seconds. When starting the invasive version with 1 process, the total runtime was 2.251.

7 Conclusion

Several benchmarks were run in order to obtain performance data related to the improvements and new features introduced with the Invasive MPI library. It was shown that the new process manager significantly reduced the latency of launching an MPI job. For the infect operation, *MPI_Init*, *fork* and *exec* took a large percentage of the total time of the single process case where minimal communication is required. This was also the case when doing the operation for the maximum case of 48 processes. It can therefore be concluded that the optimizations done in this work result in *MPI_Comm_spawn* and *MPI_Comm_infect* operations that are compute bound in the SCC. It was also shown that the invade and retreat operations are considerably faster than infect.

In general, an MPI application can be modified to be invasive by the addition of deciding criteria, a repartitioning mechanism and the use of the invasive extensions to MPI. For complex applications, for example those that rely on adaptive mesh refinement algorithms, the repartitioning strategy will be challenging; for this reason, a parallel numerical integrator was introduced as a first demostrator, since its repartition step is trivial due to the absence of data dependencies. In this case, the base parallel algorithm was a parallel mid-point based integration algorithm and the criterion was based on Simpson's rule.

The invasive programming model can be supported as extensions to MPI. This approach preserves compatibility with a large number of applications while adding the option of resource aware programming. The well designed and known features of MPI can be used together with invasive operations in parallel applications. Furthermore, improvements done to process management and library performance benefit both standard MPI and invasive applications.

Future work for the Invasive MPI Library includes the addition of more resources to be handled by the process manager. Currently, the cores and the memory available to it are invaded simultaneously; a separation can be made and the interface should allow for their invasion separately. In addition, the dynamic voltage and frequency scaling functionality of the SCC should be exploited. Finally, to ease the cost of migration, support for automatic repartitioning of data structures should be added.

Acknowledgments. We would like to thank Prof. Bungartz and Martin Schreiber, of the Chair of Scientific Computing at the TUM, for their collaboration with the numerical integrator.

References

1. Held, J.: "Single-Chip Cloud Computer", an IA Tera-Scale Research Processor. In: Guarracino, M.R., Vivien, F., Träff, J.L., Cannatoro, M., Danelutto, M., Hast, A., Perla, F., Knüpfer, A., Di Martino, B., Alexander, M. (eds.) Euro-Par-Workshop 2010. LNCS, vol. 6586, p. 85. Springer, Heidelberg (2011)
2. Mattson, T.G., Van der Wijngaart, R.F., Riepen, M., et al.: The 48-core SCC processor: The programmer's view. In: Supercomputing Conference, ACM/IEEE, New Orleans, LA (2010)
3. Clauss, C., Lankes, S., Galowicz, J., Bemmerl, T.: iRCCE: A Non-blocking Communication Extension to the RCCE Communication Library for the Intel Single-Chip Cloud Computer, Chair for Operating Systems, RWTH Aachen University (December 17, 2010)
4. Comprés Ureña, I.A., Riepen, M., Konow, M.: RCKMPI – Lightweight MPI Implementation for Intel's Single-Chip Cloud Computer (SCC). In: Cotronis, Y., Danalis, A., Nikolopoulos, D.S., Dongarra, J. (eds.) EuroMPI 2011. LNCS, vol. 6960, pp. 208–217. Springer, Heidelberg (2011)
5. van der Wijngaart, R.F., Mattson, T.G., Haas, W.: Light-weight communications on intel's single-chip cloud computer processor. SIGOPS Oper. Syst. Rev. 45, 73–83 (2011)
6. Verstraaten, M., Grelck, C., van Tol, M.W., Bakker, R., Jesshope, C.R.: On mapping distributed s-net to the 48-core intel SCC processor. In: Third MARC Symposium, Ettlingen, Germany (July 2011)
7. Teich, J., Henkel, J., Herkersdorf, A., Schmitt-Landsiedel, D., Schröder-Preikschat, W., Snelting, G.: Invasive Computing: An Overview. In: Multiprocessor System-on-Chip – Hardware Design and Tool Integration, pp. 241–268. Springer, Heidelberg (2011)
8. Clauss, C., Lankes, S., Bemmerl, T.: Performance Tuning of SCC-MPICH by Means of the Proposed MPI-3.0 Tool Interface. In: Cotronis, Y., Danalis, A., Nikolopoulos, D.S., Dongarra, J. (eds.) EuroMPI 2011. LNCS, vol. 6960, pp. 318–320. Springer, Heidelberg (2011)
9. McKeeman, W.M.: Algorithm 145: Adaptive numerical integration by simpson's rule. Commun. ACM 5, 603–604 (1962)
10. Khan, R.: SCC Baremetal Framework Bandwidth and Power Findings. In: 2nd Many-core Architecture Research Community (MARC) Symposium, Santa Clara (March 30, 2011), (presentation) `http://communities.intel.com/docs/DOC-6258`
11. MPICH2: High-performance and Widely Portable MPI, `http://www.mcs.anl.gov/research/projects/mpich2/`
12. Transregional Research Center InvasIC, `http://www.invasic.de`