Auto Source Code Generation and Run-Time Infrastructure and Environment for High Performance, Distributed Computing Systems

Minesh I. Patel Ph.D.[1], Karl Jordan[1], Mattew Clark Ph.D.[1],
and Devesh Bhatt Ph.D.

[1] Honeywell Space Systems-Commercial Systems Operations,
13350 U.S. Highway 19 North,
Clearwater, Florida, USA, 33764
{minesh.patel, karl.l.jordan, mathew.clark}@honeywell.com
[2] Honeywell Technology Center,
Minneapolis, Minnesota,
devesh.bhatt@honeywell.com

Abstract. With the emergence of inexpensive commercial off-the-shelf (COTS) parts, heterogeneous multi-processor HPC platforms have now become more affordable. However, the effort required in developing real-time applications that require high-performance and high input/output bandwidth for the HPC systems is still difficult. Honeywell Inc. has released a suite of tools called the Systems and Applications Genesis Environment (SAGE) which allows an engineer to develop and field applications efficiently on the HPCs. This paper briefly describes the SAGE tool suite, which is followed by a detailed description of the SAGE automatic code generation and run-time components used for COTS based heterogeneous HPC platform. Experiments were conducted and demonstrated to show that the SAGE generated glue (source) code with run-time executes comparably or within 75% efficiency to hand coded version of the Parallel 2D FFT and Distributed Corner Turn benchmarks that were executed on CSPI, Mercury and SKY compute platforms.

1 Introduction

Many Military, Industrial and Commercial systems require real-time, high-performance and high input/output bandwidth performance. Such applications include radar, signal and image processing, computer vision, pattern recognition, real time controls and optimization. The complexities of high performance computing (HPC) resources have made it difficult to port and fully implement the various applications. With the availability of inexpensive HPC systems based on commercial hardware, the high demands of military and industrial applications can be met. However, the potential benefit of using high performance parallel hardware is offset by the effort required to develop the application. Honeywell Inc. has release a set of user friendly tools that

offer the application and systems engineer ways to use the computing resources for application development. By tuning processes, improving application efficiency and throughput, and automatic mapping, partitioning and glue (source) code generation, the engineer can improve productivity and turn around time, and lower development cost.

This paper describes the Systems and Applications Genesis Environment (SAGE) and its auto-glue (source) code generation and run-time components. We first provide a brief overview of Honeywell's SAGE tool suite. This is followed by a description of the SAGE's auto glue code generation and run-time components. Finally, the experiments and results describing the comparison between the performance of the auto-generated glue code and hand-coded benchmarking applications, the Parallel 2D-FFT and the distributed corner turn is provided.

1.1 Systems and Applications Genesis Environment (SAGE)

Honeywell has developed an integrated tool suite for system design called the Systems and Applications Genesis Environment (SAGE)[1]. The tool suite provides complete lifecycle development through an integrated combination of tools potentially reducing design and development costs. The SAGE approach to application development is to bring together under a common GUI, a set of collaborating tools designed specifically for each phase of a system's development lifecycle. SAGE consists of the SAGE: Designer, the SAGE: Architecture Trades and Optimization Tool (AToT) and the SAGE: Visualizer.

Typically the design process begins with the Designer. The engineer can use the Designer to describe and capture the hardware and software/application architectures of the system and the mapping between application to hardware, which may be refined or narrowed by AToT. In the Designer, application/system and hardware co-design can be performed using the Designer's three editors, the application editor, data type editor and the hardware editor. The application editor is used to build a graphical view or model of the application by connecting functional or behavioral blocks (hierarchical) in a data flow manner through user defined or COTS functional libraries. The data type editor is used to define the various data types and striping and parallelization relationships for the different functions in the application editor. In the hardware editor, the hardware architecture is built hierarchically from the processor all the way up to the system level. All primitive and hierarchical blocks are stored on software and hardware "shelves" for later reuse. Items on the hardware shelf include workstations, other embedded computers, CPU chips, memory, ASICs, FPGAs, etc. The application and system designs can be refined using the software shelf items such as other COTS functional or user defined blocks. The entire software development environment integrates COTS-supplied components (compilers and run-time system, and libraries), along with custom, user-supplied software and hardware components (application code, libraries, etc.). Combining elements from the hardware shelf, the software shelf, and trade information, the engineer can construct an executable which maps software components onto hardware resources.

Once the performance requirements, application and hardware of the system are captured in the Designer, the information is sent to AToT. AToT will analyze and interpret the captured information, which drives optimization and trade-off activities described in the following section. After the architecture trades process has determined a target hardware architecture, the genetic algorithm based partitioning and mapping capability of AToT assigns the application tasks to the multi-processor, heterogeneous architecture. AToT can be employed for total design optimization, which includes load balancing of CPU resources, optimizing over latency constraints, communication minimization and scheduling of CPUs and busses.

When all the details of the system design have been made, the engineer may instrument and auto-generate the actual application code, which can be compiled and executed on certain supported testbed platforms. The SAGE Visualizer is a configurable instrumentation package that enables the designer to visualize the execution of the application through a variety of graphical displays that are fed by probes placed within the generated code. The Visualizer allows the designer to configure the instrumentation probes to measure application performance, and search for problems in the system, such as bottlenecks or violated latency thresholds.

2 Auto-Glue Code Generation and Run-Time Kernel

The SAGE glue-code generator is implemented in Alter, a programming language similar to Lisp in its syntax and style, which provides a direct interface to the contents of a SAGE model. Alter is designed to enable the tool developer to traverse the objects and arc connections in a model, collect the relevant information from the various attributes and properties, and then output the information in a particular format for the application. In the context of the glue-code generator, Alter traverses through the SAGE model and generates source code that can be compiled with application function libraries and the SAGE run-time as shown in Figure 1. The basic Alter language provides the constructs to perform the traditional programming tasks, such as procedure encapsulation, conditionals, looping, variable declaration, and recursion. The language also includes a set of standard calls to access certain features in SAGE, such as setting or retrieving a property value from an object.

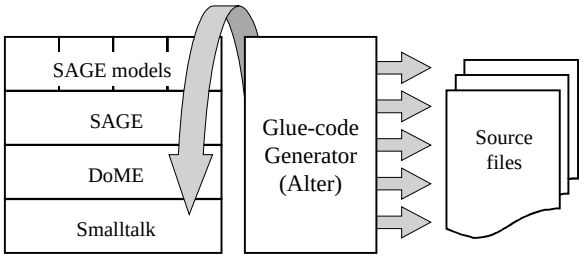

Figure 1.0 The SAGE glue-code generator gains access into the internal SAGE design tool environment, traverses objects in the models to filter relevant information, and then outputs the information in formats particular to the SAGE run-time source files. The SAGE glue-code generator is implemented in Alter, the programming language that facilitates the traversal and manipulation of DoME-based objects and graphs.

The SAGE run-time kernel is responsible for all sequencing of functions, data striping, and buffer management. To better cover the wide range of application domains, it is necessary to capture the notation of complex data distribution between functional software modules. In the data-flow programming model of the SAGE design notation, this requirement is handled by the port striping features. In short, the port striping conventions enable the system designer to define complex data distribution patterns between functions in a multi-threaded environment. A function's port object is the sending and receiving point for all data-flow communication between functions; the striping characteristics of a data-flow connection are defined on the source and destination ports. As mentioned previously, the glue-code generator develops several SAGE run-time source files, using information generated from the application model. For example, the function table is generated from a list of all function instances in the SAGE design. SAGE Designer orders all function instances and assigns them IDs from 0.. N - 1. The SAGE runtime executes functions based on this ID, which is the index of this descriptor into the function table. Similarly, information is extracted from the model that allows the runtime to perform data striping.

A function port can be defined in the model to be of type **replicated** or **striped**. Replicated ports represent data-flow communications in which the data is replicated for each thread of the host function. Striped ports represent data-flow communications in which the data is sliced or divided evenly among the threads of the host function. The port striping type applies to both sending (outgoing) and receiving (incoming) ports.

The runtime is responsible for striping the data based on the model information specified in the glue-code. It performs this operation using data buffers. Located and shared between each port on the sender and receiver functions is the SAGE notion of a *logical buffer*. A logical buffer is a logical representation of the data flow between sender and receiver function threads. It contains the striding information, total buffer size (before striding), thread information (number and type), etc. The logical buffer is defined by the glue-code using the application model's properties. The runtime uses the logical buffer and the striding information to create physical buffers for message transfer.

3 Experiments

In our experiments, we intend to show that SAGE produces executable code that is comparable to hand generated code for the targeted high performance computing platform for a selected set of benchmark applications. It is understood that tools which

can auto generate code that can surpass performance wise hand coded application implementations is still work to be done. It is our intention to show that an application or system engineer can develop an application (conceptual, first cut or final version) using SAGE quickly and that the resulting solution is comparable both in performance and code size to hand coded versions. Additionally, the application can be refined for better performance by using the SAGE visualization software and by adding hand tuned functions to the SAGE reuse library for the target hardware platform.

3.1 Benchmark Applications

The benchmark applications chosen are algorithms that have been used by Rome Laboratories and MITRE in their benchmarking efforts of COTS based high performance computing systems such as from Mercury, Sky and CSPI. The applications chosen for our experiments are the parallel 2D FFT and the parallel distributed Corner Turn executing on a 1024x1024 data matrix. The two applications and data set were provided by CSPI. Performance results of the two applications executing on a Mercury, CSPI, SIGI and SKY platforms were obtained from MITRE[2]. For each of the hardware platforms, MITRE performed measurements using several node configurations (node counts). Additionally, high performance-computing vendors developed their own MPI implementation optimized for their hardware. The traditional MPI implementation have a built in function for performing the corner turn operation, namely the MPI_All_to_All function, each vendor implemented their own version tailored to their respective hardware for the most optimal performance.

3.2 Target Machine

The target hardware platform for performing the SAGE glue code and run-time experiments was chosen to be a 200 MHz Power PC 603e based high performance computing system provided by CSPI. The target system contained two quad-Power PC boards with the VxWorks operating system housed within a 21 Slot VME chassis. Each Power PC has 64 Mbytes of DRAM and can communicate through 160 MBytes Myrinet fabric interconnect to each other (intra-board) and to the outside world (inter-board). CSPI also provided all software including the VxWorks operating system, MPI implementation and the CSPI ISSPL functional libraries. As part of the Honeywell IR&D program and corporate alliance with CSPI, the SAGE tool was ported to CSPI target hardware platform. The term “port” corresponds to the capturing of all knowledge associated with programming to the CSPI hardware. Such knowledge that is captures includes the ISSPL function libraries on to the appropriate shelves, the CSPI board specific run-time software and programming methodology. It is expected that within the year, additional hardware platforms will be folded into the SAGE knowledge repository. It should be noted that SAGE hides the complexities of programming to COTS high performance computing hardware from the application developer. Once an application is developed, that application becomes portable to other SAGE supported platforms.

3.3 Experiments and Test Method

The experiments for the SAGE auto glue code generation and run-time components will be conducted in four steps. First, the application will be modeled using the Designer. Second, the different node configurations and mappings will be chosen through the Designer. Third, the glue code will be auto-generated where each node configuration and mapping will be executed ten times where each execution consists of a 100 iterations. The fourth step is the actual execution. The final performance number for that execution will average the 100*10 results into a final average result. When results are reported, a period is defined to be the time between input data sets while latency is the time required to process a single data set. The latency corresponds to the time from when the first data leaves the data source to the time the final result is output to the data sink.

3.4 Results

The results of the experiments are shown in Table 1.0. Table 1.0 shows the actual performance numbers for the two benchmark applications executing on 4 and 8 node configurations with 256, 512 and 1024 data sets. Each entry denotes the average of the 10*100 executions with cumulative averages shown in the last column. The table shows that the SAGE auto-generated code executed within an average 86% of the hand-coded versions on the CSPI hardware. For the distributed corner turn, the SAGE generated code running on the CSPI platform performed as well as the hand-coded CSPI version with an average overhead of 20%. For the 2D FFT, SAGE showed, on average, 17% cost in overhead.

Table 1.0 Comparison of hand-coded and auto-generated code for CSPI

Application	Array Size	Number of Processing Nodes						Average
		CSPI Hand Coded		SAGE AutoGen		% of Hand Coded		
		4	8	4	8	4	8	
2D FFT	256 x 256	14.8	8.496	15.8	9.4	93.7	90.4	92.0
	512 x 512	63.77	33.902	70.22	37.75	90.8	89.8	90.3
	1024 x 1024	267	137	312	169	85.6	81.1	83.3
Corner Turn	256 x 256	6.68	4.27	7.786	4.753	85.8	89.8	87.8
	512 x 512							
	1024 x 1024	86.53	52.2	108.822	65.135	79.5	80.1	79.8
								86.7

For the distributed corner turn, the SAGE generated code running on the CSPI platform performed as well as the hand coded CSPI version with on average 25% cost in overhead. A performance hit was taken on a two-node configuration. Here, the SAGE run-time buffer management scheme assigns unique logical buffers to the data

per function which can cause extra data access times when compared to the CSPI implementation. For the 2D FFT, SAGE showed on average 20% cost in overhead.

4 Conclusions

The SAGE tool suite provides a powerful graphical and interactive interface for the creation of executable systems and applications based on customer defined specifications with fewer errors and an order of magnitude reduction in development time. The SAGE auto glue code generation and run-time components delivered and executed the two benchmark applications at 77.5 % of hand code versions. Although the performance of the auto-generated code is not equal to the hand code versions, tools that can generate such code are many years away. Work is currently underway to improve the performance of the glue code generation component that will reach levels of 90% of hand coded performance. The use of SAGE provides the application or systems engineer a way to rapidly develop an application on the target system with reasonable assurances that the performance of the auto-generated code for the application will not be magnitudes different from hand coded versions. And since the current SAGE tool makes the target system transparent to the engineer, the application developed is portable to other SAGE supported hardware platforms. The designer simply needs to re-generate the glue code for the new hardware platform. The time saved by using SAGE can now be more effectively used to perform such tasks as improving the applications performance on the current hardware platform, trading and testing the application on other hardware platforms, and moving on to the next project.

References

1. Honeywell's Systems and Applications Genesis Environment (SAGE™) Product Line, http://www.honeywell.com/sage.
2. Games, Richard, "Cross-Vendor Parallel Performance," Slides Taken from: Real-Time Embedded High Performance Computing State-of-the-Art, MITRE Corporation, Presented at DARPA Embedded Systems PI Meeting, Maui, Hawaii, March 16, 1999.