Benchmarks for the GINA platform

SPLASH2 and NPB experiments
March 3, 2008

Giovanni Beltrame
Luca Fossati

Contents

1 Synopsis

This document describes the execution of parallel benchmarks on the GINA platform in order to establish a term of reference for ESA next-generation microprocessor. The main idea behind this study is that processing power does not scale uniformly with the number of cores used, due to concurrent access to shared resources. By verifying and measuring how parallelized software scales on a given architecture, it is possible to determine its shortcomings and ways to improve its scalability.

In general, shared-memory parallel software uses a *programming model* that specifies the primitives to create, synchronize and destroy threads of execution. In this work we focus on two popular programming models, *PARMACS* and *OpenMP*, as most free and commercial benchmarks rely on them. Both programming models present an API to the programmer, the former as macros, the latter as compiler pragmas.

The two models are general, they are not targeted to any particular architecture, thus they need a "back-end" implementation to work on a specific system. In this work we use the POSIX Threads (generally referred to as *pthreads*) API for the back-end, due to its diffusion and support. Pthread implementations of both PARMACS and OpenMP are available as open-source.

Working with POSIX Threads implies using of a shared-memory multiprocessor POSIX operating system. As *eCos* has these characteristics and as it has already been ported to GINA, it was used as the choice operating system.

The performance of the benchmarks was evaluated using the board timers to compute the timing and TSIM to determine the number of instructions involved in the core of each benchmark. Thread management, inter-processor interrupts, etc. are considered as overhead, and they are not included in the instruction count. The result is a number of instructions and an execution time per benchmark, giving an overall Millions Operations Per Second (MOPS) figure; this figures includes both integer and floating point operations.

In the following, we describe the setup for both the benchmarks and eCos, the execution procesure, and we discuss the results.

2 eCos Setup

eCos is a highly configurable, open-source embedded operating system. Its sources (but not its configuration) were provided with GINA CDs. A vanilla configuration had to be extracted from the sources, and then adapted for the use of benchmarks. This involved a few trials, but can be summarized as:

- Activation of FPU, SMP, and hardware serial drivers support
- Installation of the POSIX compatibility layer

It is worth noting how the sources on the CD are not the same sources used to build the sample binaries, as they have a different memory map. This required some additional patching of eCos' sources, moving the application entry point from 0x40000000 (SRAM) to 0x60000000 (SDRAM), in order to obtain the same memory map of the sample binaries provided on the CDs.

eCos was compiled using the gcc-3.4.4 compiler from Gaisler website, and all its test run as expected. Compiling the examples with gcc-4.2.2 gave some minor performance increase in the samples. However, when eCos itself was compiled with gcc-4.2.2, curiously the system didn't boot (independently of the optimization flags used).

As an additional note, the dhrystone example (provided on the GINA CDs) used a 100 divider for the clock tick counter to measure time, but GINA's clock in its current implementation is 40MHz, giving the wrong MIPS calculation (overestimated by 60%).

3 SPLASH2

The SPLASH2 set of benchmarks was developed by Stanford in 1995 to test parallel supercomputer architectures. As modern embedded systems are much more powerful and sport multi-core architectures, this set can be used with little adaptation. The benchmarks are divided in two sets: applications and kernels, applications being somewhat larger and more complex (and generally slightly less parallel). The SPLASH2 benchmarks are:

BARNES : this application implements the Barnes-Hut method to simulate the interaction of a system of bodies (N-body problem). This application is computationally but not memory intensive.

FMM : this application implements a parallel adaptive Fast Multipole Method to simulate the interaction of a system of bodies (N-body problem).

OCEAN : this program simulates large-scale ocean movements based on eddy and boundary currents. Two implementations are provided in the SPLASH-2 distribution:

- *Non-contiguous partition allocation*: this implementation uses two-dimensional arrays. This data structure prevents partitions from being allocated contiguously, but leads to a conceptually simple programming implementation.
- *Contiguous partition allocation*: this implementation uses 3-D arrays. The first dimension specifies the processor which owns the partition, and the second and third dimensions specify the x and y offset within a partition. This data structure should enhance data locality properties (so it should lead to a better exploitation of the on-chip caches).

RADIOSITY : this code computes the equilibrium distribution of light in a scene using the hierarchical diffuse radiosity method. This benchmark is both CPU and memory intensive.

RAYTRACE : This code renders a three-dimensional scene onto a two-dimensional image plane using optimized ray tracing. A hierarchical uniform grid is used to represent the scene for efficient access, and early ray termination and antialiasing are implemented. The complexity and the use of large input files and additional libraries prevented the use of this benchmark.

VOLREND : This code renders a three-dimensional volume onto a two-dimensional image plane using an optimized ray casting technique developed by Marc Levoy. A hierarchical octree data structure is used to represent the scene for efficient access, and early ray termination and antialiasing are implemented. The same considerations for raytrace apply here.

WATER-SPATIAL : This code solves molecular dynamics N-body problem of water. It imposes a 3-d spatial data structure on the cubical domain, resulting in a 3-d grid of boxes. This benchmark is both CPU and memory intensive as boxes are stored as linked lists.

WATER-NSQUARED : similar to water-spatial, but using a different algorithm, less memory intensive.

CHOLESKY : this kernel performs blocked Cholesky Factorization on a sparse matrix. The matrix definition is stored on a large file, and so the benchmark was not included.

FFT : this kernel is a complex, one-dimensional version of the "Six-Step" FFT. CPU intensive.

LU : this kernel factors a dense matrix into the product of a lower triangular and an upper triangular matrix. Two implementations are provided (as for OCEAN), with different memory access strategies.

RADIX : this kernel implements an integer radix sort. This is the least demanding of the benchmarks concerning CPU power and memory bandwidth.

All these benchmarks use the PARMACS programming model. In this work, we used a freely available pthread PARMACS implementation, patched in order to provide a correct timing figures on the GINA board. Although Gaisler patches to eCos provided a PARMACS implementation that relies on eCos threads instead of the POSIX compatibility layer, these were not used as we wanted to estimate the overhead of the standard POSIX programming model. Gaisler-provided eCos configuration files did not feature the required POSIX compatibility layer, so, as previously stated, this feature had to be added to the configuration. A different specific configuration for eCos was also defined to run the benchmarks on TSIM; this step was necessary in order to obtain the number of instructions involved in the core routines of each benchmark (initialization, generally single threaded, is excluded from this count).

As TSIM-LEON3 was not available, TSIM-LEON was used, and its configuration parameters were tweaked to mimick as much as possible GINA configuration: RAM, SDRAM, caches are the same, with the exception of the cache line size. This does not affect the instruction count in any case. The benchmarking was structured as:

- One run of the benchmark on the host machine to test the PARMAC macros using pthreads
- One TSIM simulation using hardware breakpoints around the core of the benchmark to determine the instruction count
- Three simulations of the benchmark on GINA, using one, two and four cores respectively timing the benchmarks at the same breakpoints as used in TSIM

Unfortunately, not all the benchmarks run as expected: we experienced some unexpected crashes on the board and on TSIM although all of the benchmarks ran correctly locally on Linux i686. Changing the PARMACS macros, using another barrier style, reduced the number of crashes but it didn't really solve the problem. The benchmarks that failed are:

- OCEAN, using the contiguous-sets implementation. Although OCEAN ran on TSIM, it kept hanging on GINA, even when using only one thread, (this excludes synchronization or coherency problems). Little changes of the executable changed the way the program crashed, leading us to think of some pointer-related issue that however remains unfound
- BARNES: the application runs smoothly on TSIM but crashes during initialization (when accessing a vector in particular). It is not clear what causes the IU to enter error mode (write error, code $0x2b$)
- FMM: the benchmarks works on TSIM, but stops at a non-existent watchpoint (perhaps an unwanted write to the DSU?) as soon as the calculations start
- RADIOSITY: apparently it gets locked in an infinite loop on both TSIM and the GINA board when using pthreads. It works on TSIM only when run without threading

The results for the remaining benchmarks are outlined in Table 1, with the last line showing the adjusted average, excluding outliers (the fastest and the slowest benchmark). Figure 1 shows in graphical form the MOPS trend as the number

Table 1: Performance of the GR-CPCI-XC4V board configured with GINA, scaled to 266MHz

Benchmark	Bus b/w [%]	1 processor speed (MOPS)	2 processors speed (MOPS)	4 processors speed (MOPS)
FFT	16.9	172.96	320.59	461.57
LU	14.4	78.67	146.54	195.16
LU-NC	26.7	61.84	101.68	106.09
RADIX	6.3	105.73	168.11	164.81
OCEAN	15.3	92.86	202.03	230.36
WATER-N	9.0	68.56	125.66	201.7
WATER-S	9.1	76.49	133.01	221.01
Average	13.9	93.87	171.09	225.81
Average adj.	10.82	84.46	148.69	200.48

of cores used increases, together with adjusted standard deviation (thick gray bar) and minumum and maximum error (thin line error bar). On average, the trend shows diminishing returns, roughly 1.76x and 1.34x performance when going from 1 to 2 and 2 to 4 cores, respectively. The thick gray bar represents the standard deviation of the benchmarks when the two outliers at the extemes of the error bar are removed, showing a consistent behavior of the benchmarks, excluding FFT (highly parallelizable and scalable) and LU-NC (memory access inefficient). Another interesting result is produced by correlating the single-core bandwidth utilization of the benchmarks with the speedup obtained with 2 and 4 cores, as shown in Table 2. The Pearson correlation coefficient RHO (we

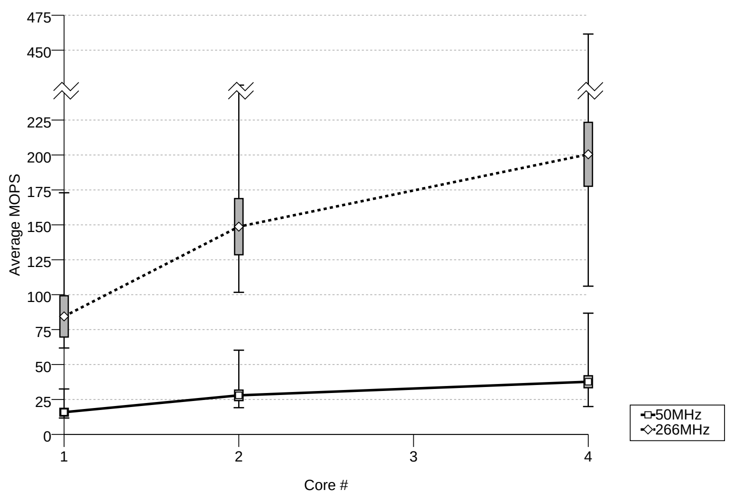

Figure 1: MOPS trend for the SPLASH2 benchmarks

Table 2: Speedup and single bandwidth utilization correlation

	Bandwidth	Speedup 2	Speedup 4
RADIX	6.3	1.59	0.98
WATER-S	9	1.74	1.66
WATER-N	9.1	1.83	1.61
LU	14.4	1.86	1.33
OCEAN-NC	15.3	1.83	1.29
FFT	16.9	1.85	1.44
LU-NC	26.7	1.64	1.04
RHO (Pearson)	-	-0.00094	-0.3467

used Pearson as the relation between speedup and *doubling* of cores is almost linear) shows that the bandwidth utilization is totally uncorrelated with the speedup obtained with 2 cores: this agrees with the fact that the bus is not close to saturation. The correlation increases as we go to 4 cores, showing some initial saturation effects, but remaining low and with some evident outliars, like RADIX where saturation has no impact. It is worth noting that CPU use is 100% in all benchmarks (they are all computationlly intensive with no I/O operations).

4 NASA Parallel Benchmarks

A more interesting and more self-contained benchmark suite is the NASA Parallel Benchmarks (NPB). These benchmarks are based on the OpenMP programming model. The NPB suite is composed by:

BT is a simulated CFD (Computational Fluid Dynamics) application that uses an implicit algorithm to solve 3-dimensional (3- D) compressible Navier-Stokes equations. The finite differences solution to the problem is based on an Alternating Direction Implicit (ADI) approximate factorization that decouples the x, y and z dimensions. The resulting systems are Block-Tridiagonal of 5x5 blocks and are solved sequentially along each dimension.

SP is a simulated CFD application that has a similar structure to BT. The finite differences solution to the problem is based on a Beam-Warming approximate factorization that decouples the x, y and z dimensions. The resulting system has Scalar Pentadiagonal bands of linear equations that are solved sequentially along each dimension.

LU is a simulated CFD application that uses symmetric successive over-relaxation (SSOR) method to solve a seven-block-diagonal system resulting from finite-difference discretization of the Navier-Stokes equations in 3-D by splitting it into block Lower and Upper triangular systems.

FT contains the computational kernel of a 3-D fast Fourier Transform (FFT)-based spectral method. FT performs three one-dimensional (1-D) FFT's, one for each dimension.

MG uses a V-cycle MultiGrid method to compute the solution of the 3-D scalar Poisson equation. The algorithm works continuously on a set of grids that are made between coarse and fine. It tests both short and long distance data movement.

CG uses a Conjugate Gradient method to compute an approximation to the smallest eigenvalue of a large, sparse, unstructured matrix. This kernel tests unstructured grid computations and communications by using a matrix with randomly generated locations of entries.

EP is an Embarrassingly Parallel benchmark. It generates pairs of Gaussian random deviates according to a specific scheme. The goal is to establish the reference point for peak performance of a given platform.

A compatible compiler is needed to compile OpenMP programs. In this work, we tested three possible solutions:

- GCC-4.2: the newest version of GCC natively supports OpenMP, using a runtime library called libgomp. As programs for eCos are compiled with some special parameters, the runtime libgomp has to be compiled with different flags with respect to the rest of the cross-compiler. GCC does not allow such a setup, and the libgomp had to be extracted manually, compiled outside the tree, and linked to the test executables. The process required some tweaking but compiled without problems in the end

- Omni: an OpenMP source-to-source compiler that transforms OpenMP code in C/C++, relying on a proper C/C++ compiler for the creation of executables. Unfortunately, Omni relies on an undocumented runtime library, and does not accept many of the C constructs used in eCos. Even though the NPB suite compiled (but not linked), the cross-compilation of the runtime library was not taken in consideration

- OMPI: similar to Omni, but more compact, with better structure, and a very simple runtime library creation mechanism. A special runtime for eCos and GINA was created, and successfully linked to the NPB benchmarks. eCos includes needed some tweaking as they used non-standard C constructs that OMPI didn't recognize

All the benchmarks were tested on GINA and TSIM, but none of them worked properly. We suspect this is due to the posix-based runtime libraries, as the benchmarks start but crash in different ways as threading begins. Many workarounds have been tried (increasing stack size, changing the malloc strategy, etc.), to no avail. Notice how eCos configuration had to be modified (specifying that static constructors had to be executed in the main thread context) in order for the benchmark to boot. It is worth noting that the NPB were successfully run on the ReSP simulation framework with the same libgomp configuration and a closely resembling eCos configuration, but using four ARM7 processors.

5 Conclusions

A first analysis of the results leads to the following conclusions:

1. Increasing the number of cores from 2 to 4 does not scale over 1.4x performance for the SPLASH2 benchmarks. This shows that the current GINA architecture using eCos SMP (or the benchmarks) are reaching the scalability limit. Further experimentation using an emulated POSIX layer in the simulator will allow us to separate the OS overhead from the architectural scalability limit

2. The absolute value of 220 MOPS at 266MHz is a good indication of what might be the next generation of this kind of processor. Expecting a frequency of 300MHz and a more efficient communication mechanism, it is reasonable to target 400 MOPS on average

3. eCos POSIX threading makes the implementation of OpenMP very difficult, unless a proper runtime is developed. In general, eCos non-standard coding represents an issue for its future use

4. The same benchmarking can be applied using MPI benchmarks instead of OpenMP, drawing some interesting consideration on the use of message passing as opposed to shared memory (this requires message passing support in the Operating System)