Some Properties of Selected Parallel Architectures

André Rigland Brodtkorb and Trond Runar Hagen

SINTEF ICT, Department of Applied Mathematics,
P.O. Box 124, Blindern, N-0314 Oslo, Norway
Email: {Andre.Brodtkorb,Trond.R.Hagen}@sintef.no

Abstract. In this paper we explore three widespread parallel architectures: the Graphics Processing Unit (GPU), the Cell BE processor, and multi-core CPUs. We have implemented four algorithms on these three architectures: the heat equation, inpainting using the heat equation, computing the Mandelbrot set, and MJPEG movie compression. We use these four algorithms to exemplify the benefits and drawbacks of each parallel architecture.

1 Introduction

In recent years, we have seen an explosion in the use of commodity level parallel architectures. We examine three such parallel architectures: multi-core CPUs using OpenMP, Cell BE using libspe 2, and the graphics processing unit (GPU) using NVIDIA CUDA.

Our first example of a prallel architecture, multi-core CPUs consists of multiple general-purpose cores on a single chip. These cores have a large cache, and a lot of logic for instruction level parallelism (ILP). Most of the chip area is typically used for cache and logic, leaving a relatively small number of transistors for pure arithmetic calculations. Using OpenMP, an API for multi-threading in a shared memory system, the programmer identifies parallel portions of the program, and instructs the compiler to perform parallelization.

The Cell BE is a heterogeneous processing unit, consisting of one general purpose processor, the power processor element (PPE), and eight accelerator cores, the synergistic processing elements (SPEs). The SPEs have a small local store to hold their program and data, very little logic for ILP, and are connected to the PPE through a fast on-chip interconnect. The API used to access and program the SPEs is libspe 2. Using libspe 2, a Cell BE program typically uses the PPE to load a program into each of the SPEs. The SPEs are then instantiated, start computing, and only communicate with the PPE for support functions.

The main use of GPUs is to render graphical primitives in games. GPUs consist of hundreds of stream processors where almost all transistors are used for arithmetics. The GPUs from NVIDIA can be programmed using NVIDIA CUDA, a stream processing language that views the GPU as a stream processor instead of a specialized graphics device.

2 Algorithms

We have implemented four algorithms on the parallel architectures multi-core CPU, Cell BE and GPU: the heat equation, inpainting using the heat equation, computing the Mandelbrot set, and MJPEG movie compression. The four algorithms have different memory and computational patterns, revealing architectural differences.

2.1 The Heat Equation

The heat equation describes how heat dissipates in a medium. In 2D it can be written as $u_t = k \triangle u$, where k is a material specific constant. Using an explicit finite difference scheme, the unknown solution $u_{i,j}^n$ in grid point (ih, jh) at time $(n+1) \triangle t$ is given by

$$u_{i,j}^{n+1} = k\frac{\triangle t}{h^2}\left(u_{i-1,j}^n + u_{i+1,j}^n + 4u_{i,j}^n + u_{i,j-1}^n + u_{i,j+1}^n\right). \tag{1}$$

2.2 Inpainting Using the Heat Equation

Noisy images (i.e., from analogue satellite television reception) can be repaired by inpainting the noisy images. We have used the heat equation on masked areas as a naïve example of inpainting. Using this approach, information from the area surrounding a block of noisy pixels will be diffused into the block and fill in the missing values. Technically, each noisy pixel is updated using (1), whereas the pixel $u_{i,j}^{n+1} = u_{i,j}^n$ for pixels without noise.

2.3 The Mandelbrot Set

The Mandelbrot set is a fractal which has a very simple recursive definition:

$$M = \left\{ C \in \mathbb{C} : z_0 = C, \quad z_{n+1} = z_n^2 + C, \quad \sup_{n\in\mathbb{N}} |z_n| < \infty \right\} \tag{2}$$

Informally, it is the set of all complex numbers that do not tend towards infinity when computing z_{n+1}. When computing the Mandelbrot set, one uses the fact that C belongs to M if and only if $|z_n| < 2$ for all $n \geq 0$. Typically, one picks a fixed m, and then the point C is assumed to be in the set if $|z_n| < 2$ and $n \leq m$.

2.4 MJPEG

MJPEG is "an industry standard" for compression of a movie stream. The main part of the algorithm consists of dividing each frame into 8×8 blocks, computing

the discrete cosine transform and then quantizing each block:

$$p_{u,v} = \alpha(u)\alpha(v) \sum_{i=0}^{7} \sum_{j=0}^{7} g_{i,j} \cos\left[\frac{\pi}{8}(i+0.5)\,u\right] \cos\left[\frac{\pi}{8}(j+0.5)\,v\right]$$

$$\alpha(n) = \begin{cases} \sqrt{1/8}\,, n = 0 \\ \sqrt{2/8}\,, n \neq 0 \end{cases}$$

$$r_{u,v} = \text{round}\left(p_{u,v}/q_{u,v}\right).$$

Here, $g_{i,j}$ is the element (i, j) from the 8×8 block, $p_{u,v}$ is the frequency (u, v), $q_{u,v}$ is element (u, v) of the quantization matrix, and $r_{u,v}$ is the result.

3 Results and Conclusions

We have benchmarked our algorithms on three different systems. Our CPU implementations were run on an Intel Core 2 Duo 2.4GHz with 4MiB cache. This processor has a theoretical peak performance of about 25 GFLOPS. The GPU realizations were benchmarked on an NVIDIA GeForce 8800 GTX with 768MiB GDDR3 memory, and a theoretical peak performance of around 520 GFLOPS. Our Cell BE is the 3.2GHz version with six available SPEs found in the PlayStation 3. This processor has a theoretical peak performance of around 150 GFLOPS for the SPEs.

3.1 Results

Our discretization of the heat equation leads to a formula that requires five memory reads, six floating point operations, and one memory write per element. All elements are independent, and can be computed simultaneously. Figure 1 shows the runtime of the algorithms. The graph clearly shows that it pays off to use parallel architectures, where the GPU is about 10 times faster than the single-threaded CPU implementation. The Cell BE is about 3 times faster, and the multi-threaded CPU version is about 1.7 times faster. We have a bandwidth limited application because of the memory access to arithmetic instruction ratio. This is the reason why there is a relatively small gap between the CPU and Cell BE/GPU implementations, as we are unable to utilize the computational power efficiently.

The inpainting algorithm is based on the heat equation, but now only runs on a subset of the domain. We have benchmarked using a noise mask to inpaint approximately 5% of the image. Our results show that all but the GPU run faster compared to the heat equation. This algorithm requires approximately 95% less arithmetic operations, compared to the heat equation. The GPU executes 32 elements in a synchronous SIMD fashion, and the noise distribution, we therefore most often end up utilizing only a few of the available processors while the rest simply idle. The Cell BE and CPU do not suffer from this execution model. The Cell BE is actually faster for large domains, even though it only has about one third of the processing power.

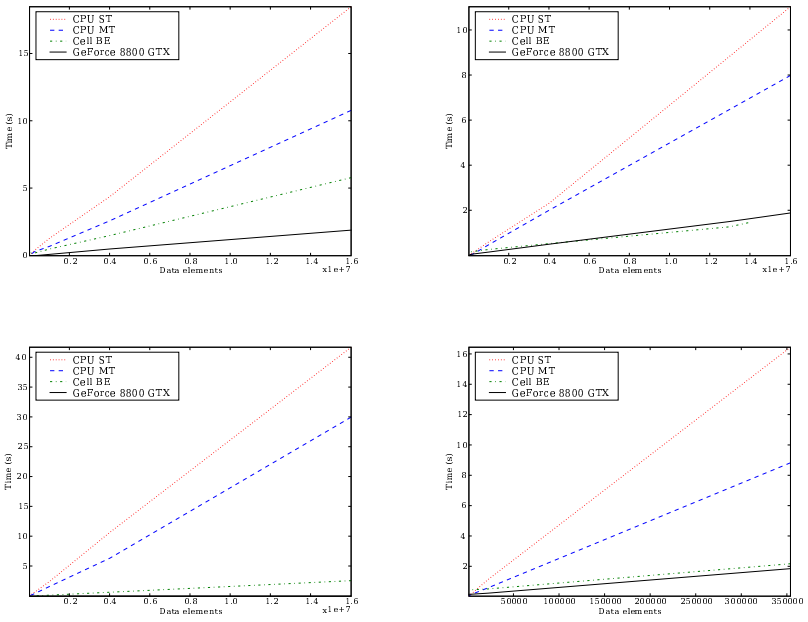

Fig. 1. Runtime of the four different algorithms: the heat equation (top left), heat inpainting (top right), the Mandelbrot set (bottom left), MJPEG (bottom right)

Computing the Mandelbrot set requires a variable number of floating point operations, and one memory write. The GPU was, by far, the fastest to compute the set, followed by the Cell BE. This is because we were able to fully utilize the available computational power, as the algorithm has a high arithmetic to memory access ratio. The Cell BE is also highly efficient because of this.

MJPEG compression requires far more floating point operations per output element compared to the heat equation. In our results, the Cell BE and GPU scale equally well, and are about 8 times faster than the single-threaded CPU. This is because we hide memory reads on the Cell BE by using double buffering, whilst we are unable to use this technique on the GPU.

3.2 Conclusions

The GPU and Cell BE implementations beat the CPU for all of our test cases. The Cell BE processor was equivalent to the GPU for our inpainting and MJPEG algorithms, even though the GPU has a much higher theoretical peak performance. The reason the Cell BE performed so well is because of its programming versatility; it is capable of advanced techniques, e.g., memory latency hiding. The GPU, on the other hand, will perform best as long as the algorithm is able to occupy all available processors on the GPU.