Using Games to Teach Parallelism to Computer Science 2nd year College Students

Deepak Vembar
Visual Computing Software Division
Intel Corporation
deepak.s.vembar@intel.com

Rowena Turner
Intel Academic Community
Intel Corporation
rowena.c.turner@intel.com

Amit Jindal
Digital Home Group
Intel Corporation
amit.jindal@intel.com

Abstract— **Parallel programming concepts are often introduced in upper-level undergraduate curriculum as a part of OS or other advanced programming courses. With the wide availability and prevalence of multi-core processors, it is imperative that students learn good programming practices to effectively use the increasing number of cores in computing environments. Parallelism concepts must be introduced sooner in the undergraduate curriculum to promote facility with parallelism and keep students competitive in today's high-tech job markets.**

We present a novel way to integrate video game based demos into sophomore computer science courses to introduce students to data parallelism and programming concepts. We have two primary objectives: update undergraduate curriculum with a laboratory-based 2nd year course on parallel computation, and introduce some of the parallelism concepts and profiling tools used in the industry.

Keywords – Parallelism, programming pedagogy, curriculum

I. INTRODUCTION

Many universities are looking for teaching methodologies to make computer science curriculum interesting to students. In particular, parallelism concepts like threading and concurrency are difficult to fully understand and are typically reserved for advanced undergraduate classes. This is problematic as the widespread adoption of parallel technology in industry requires a solid understanding of programming concepts.

Using game programming as the context within which parallelism concepts are taught is one effective way to gain student interest. Our proposed approach is novel in three ways. First, we incorporate parallel computing topics early and often into the undergraduate curriculum. Second, we present key code transitions involved in transforming a game demo from a single-threaded to a multi-threaded application using language extensions such as OpenMP or parallelism libraries Intel® Threading Building Blocks (TBB). Third, we demonstrate the benefits of using off-the-shelf profiling tools that are targeted for parallel programming as important instructional and learning aids.

II. BACKGROUND

The Visual Computing Software Enabling (VCSE) group at Intel develops demos tailored to programmers in the games industry. While some of the demos contain advanced programming concepts, others are ideal for teaching the basic tenets of parallelism, including data and task decomposition, scheduling techniques, and software architecture. These demos are designed for an audience ranging from sophomores to seniors and graduate students, with an emphasis on hands-on, laboratory instruction.

III. TEACHING PARALLELISM WITH TICKER TAPE

In this paper, we present the case study of Ticker Tape[1], a particle system simulation of falling leaves acted upon by gravity and wind effects (Figure 1). The demo is written in C++, using DirectX API. The demo starts off with a flurry of leaves falling from the sky under the influence of gravity, and the user can spawn tornados to observe the effects of wind forces on fluttering leaves as they fall to the ground. To keep the simulation simple, leaves collide with the ground, but not with each other.

The particle update loop is made up of two stages: simulation and rendering. At each time step in the simulation stage, external forces acting on the leaf (i.e. gravity and wind), are used to compute new values for position, orientation and velocity of each leaf (Figure 2). At each time step in the demo, there is repetitive work being performed on a per-leaf basis. It is important to note that as there is no inter-leaf collision in Ticker Tape, these computations are independent of each other. Once the new position and orientation of the leaves are computed, they are packaged into a vertex buffer (VB) and sent to the GPU to be rendered on the screen.

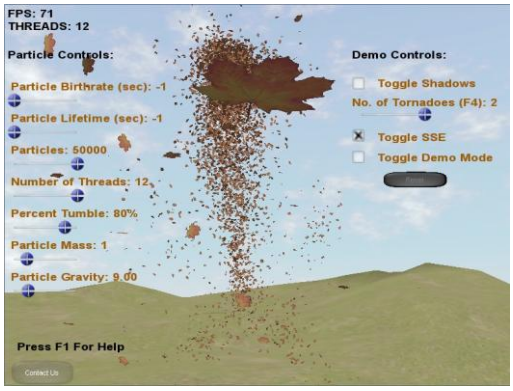

Figure 1. Screenshot of Ticker Tape with user controls to change simulation parameters.

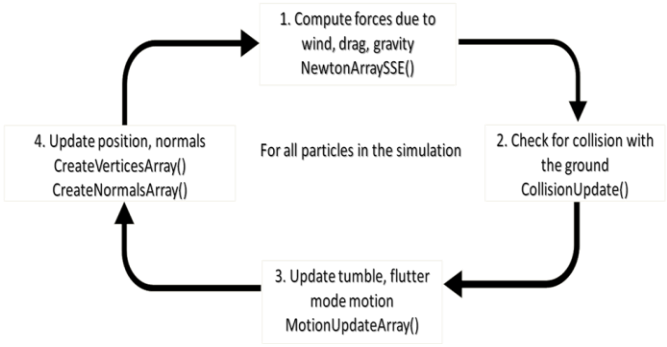

Figure 2. Overview of the simulation update stage in TickerTape

We created three different versions of the source. The first consisted of the serial version of the simulation without multi-threading. Serial code was analyzed with performance profiling tool (Intel® VTune™ Amplifier XE) to identify functions that took the longest time to complete (hotspots). The two functions that took the most time, computing new leaf positions in the file *TickerPhysicsSIMD.cpp* and filling vertex buffers for rendering on the GPU in file *TickerTape10.cpp*, were identified as good targets for optimization. As similar computation is performed on all the leaves in the simulation, code was rewritten to perform work in parallel. Parallel constructs in OpenMP were used for creating the second version, while a tasking system based on Intel® Threading Building Blocks (TBB) [2] was used to parallelize the code to create the third version of the files. Moving from the single-threaded serial version to the fully parallel, tasking with TBB version of the simulation led to a 2X speedup in the framerate (Table 1).

IV. DEVELOPMENT OF CURRICULUM MODULES

The workflow of converting Ticker Tape from a serial application to multi-threaded one was used to create lesson plans for inclusion into sophomore computer science classes. Rather than focusing on the entire codebase, key sections of Ticker Tape code as identified by VTune profiling were isolated (about 30-50 lines of code). VTune profile screenshots (Figure 3) were included in lesson slides to emphasize profiling code to identify hotspots and show

TABLE I. PERFORMANCE (MEASURED IN FRAMES PER SECOND, HIGHER IS BETTER) ON A CORE I7 980X, 4GB RAM, DISCRETE GRAPHICS CARD AND 50,000 LEAF PARTICLES FOR THE THREE VERSIONS OF CODE

Hotspot functions identified		**fps**
TickerTape10.cpp	**TickerPhysicsSIMD.cpp**	
Serial	Serial	38
Parallel with OpenMP	Parallel with OpenMP	56
Taskng with Intel TBB	Tasking with Intel TBB	77

the amount of concurrent work performed in parallel implementations of Ticker Tape. The source code of the three versions (serial, parallel with OpenMP and parallel with tasking) were used to create lab sessions and homework assignments based on the feedback received from academic collaborators. Integration into Spring 2011 sophomore classes at two schools have provided promising results. Concrete feedback is expected at the end of Spring semester, which will be used to further refine the course modules for use in Fall 2011courses.

V. CONCLUSION

We have presented a way to introduce parallel programming concepts to sophomore students via video game demos. When using TickerTape in a class or lab, students can iterate over the code doing analysis and make the changes needed to transform it from a serial to parallel application. Intel's faculty collaborators like this approach, as today's students are familiar with multimedia and rich visualization, and games act as immediately likable and interesting delivery vehicles for learning parallelism concepts at the undergraduate level.

REFERENCES

[1] Yi, M and Froemke, Q, "Ticker Tape: A Scalable 3D Particle System with Wind and Air Resistance" http://software.intel.com/en-us/articles/ticker-tape-a-scalable-3d-particle-system-with-wind-and-air-resistance/, last retrieved March, 2011.

[2] Reinders, J, "Intel Threading Building Blocks", 1st ed. O'Reilly and Associates, Inc. 2007.

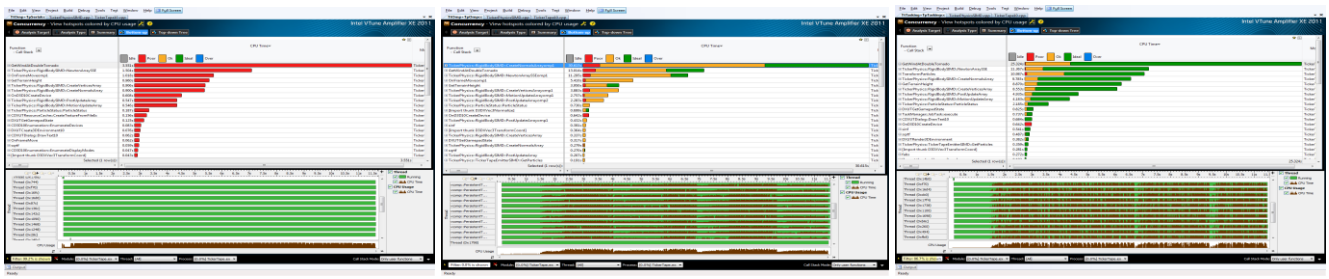

Figure 3. Output from profiling TickerTape using Intel Amplifier XE: serial (leftmost), using OpenMP(middle) and using tasking with Intel TBB(rightmost). Red bars denote that the functions are not performaning any concurrent work in parallell, while green bars denote that the there is an ideal level of concurrent work being performed