EasyPAP: a Framework for Learning Parallel Programming

Alice Lasserre, Raymond Namyst, Pierre-André Wacrenier
firstname.lastname@u-bordeaux.fr
Dept. of Computer Science
A case for a comprehensive framework

• Parallel programming is not trivial
  • Debugging is entering a world of pain
  • Understanding (bad) performance is even more challenging

• Like many teachers, we progressively added visualization facilities to our lab applications
  • Increased student’s motivation
  • Greatly helped to improve correctness

• EasyPAP goes further
  • Minimize time spent to become familiar with new problems
  • Enable quick OpenMP/MPI/OpenCL prototyping
  • Provide simple tools to analyze parallel behavior
EasyPAP: focus on parallelism!

- C library + utilities
  - Support for Pthreads, OpenMP, MPI, OpenCL
- Online rendering of 2D computations
  - Work distribution monitoring
- Trace visualization
  - Side-by-side comparison
- Plotting facilities
  - Thorough experiments & analysis
Kernels and variants

• Students are provided with sequential implementations of various kernels
  • Mandelbrot Set, Game of Life, Abelian Sandpiles, Picture Blur
  ✓ Just add a C file to create a new kernel, then compile & run

• They can design and experiment with as many variants as they can think of
  • Kernels and variants are selected on command line
  ✓ Just add a function to create a new variant, then compile & run

```c
/////////////////////////////////////////////////////////////////////// Simple sequential version (seq)
// Suggested cmdline:
// easyppap --kernel mandelset --variant seq
//
unsigned mandelset_compute_seq (unsigned nb_iter) {
    for (unsigned it = 1; it <= nb_iter; it++) {
        for (int y = 0; y < DIM; y++)
            for (int x = 0; x < DIM; x++)
                cur_img (y, x) = compute_one_pixel (y, x);
        zoom ();
    }
    return 0;
}

unsigned mandelset_compute_omp (unsigned nb_iter) { }
unsigned mandelset_compute_omp_tiled (unsigned nb_iter) { }
unsigned mandelset_compute_mpi (unsigned nb_iter) { }
```
Code instrumentation and monitoring

// Tile inner computation
static inline void do_tile (int x, int y, int width, int height, int thread)
{
    monitoring_start_tile (thread);
    for (int i = y; i < y + height; i++)
        for (int j = x; j < x + width; j++)
            cur_img (i, j) = compute_one_pixel (i, j);
    monitoring_end_tile (x, y, width, height, thread);
}

/////////////// Tiled OpenMP version (omp_tiled)
// Suggested cmdline: easyap -k mandelset -v omp_tiled -ts 32 -m
unsigned mandelset_compute_omp_tiled (unsigned nb_iter)
{
    for (unsigned it = 1; it <= nb_iter; it++) {
        #pragma omp parallel for collapse(2) schedule(runtime)
        for (int y = 0; y < DIM; y += TILE_SIZE)
            for (int x = 0; x < DIM; x += TILE_SIZE)
                do_tile (x, y, TILE_SIZE, TILE_SIZE, omp_get_thread_num ());
        zoom ();
    }
    return 0;
}
Off-line Trace Visualization

Task scheduling chart

Task-data mapping
Trace comparison

“diff” mode: iterations are re-aligned
Plotting facilities

- Experiments can easily be automated using scripts
  - No need to recompile
  - Each run records all experimental details in a CSV file

- Plotting (python) scripts
  - Ease graph selection
  - Make sure results are sound
    - Speedups automatically computed
    - Parameter consistency check
What are the main benefits?

• **Focus on parallelism**
  • Implement many variants
  • Experiments with multiple parameters

• **Quicker and deeper understanding of**
  • Scheduling
    • Load balancing, data affinity
  • Cache
    • Tiling, false sharing
  • Synchronization
    • Race conditions, barriers, task dependencies
  • Hardware specific optimizations
    • Code specialization, vectorization
EasyPAP documentation and download:

http://gforgeron.gitlab.io/easypap/