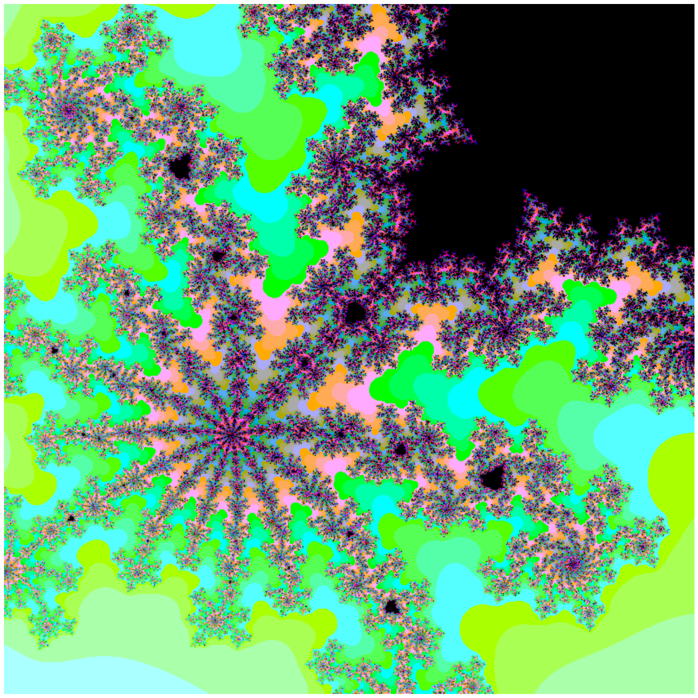
Prelude



Parallelism Review

Announcements

- The readings marked "Read:" are required.
- The readings marked "Other resources:" are NOT required reading but more for your reference.

Today

- Using OpenMP on the veges and on the Cray
- OpenMP
  - for loops
  - reductions
- Concepts: speedup, isoefficiency, critical path, work and span, etc.
- Using Tau to profile performance on the veges and the Cray

Using OpenMP on the veges (see Resources page on website)

<Demo, see class video>

Log into a vege (http://www.cs.colostate.edu/~info/machines)

Get the tar ball and unpack it

- `wget http://www.cs.colostate.edu/~cs560/Spring2012/CodeExamples/Mandel.tgz`
- `tar xzvf Mandel.tgz`

View the README file for compilation and execution directions

- `gcc –fopenmp mandel.c mytimer.c ppm.c`
- `setenv OMP_NUM_THREADS 8 (for csh and tcsh users)`
- `./a.out`

View the output

- `display mandel.ppm`

Play around with parameters

- `export OMP_NUM_THREADS=8 (for sh and bash users)`
- `./a.out -1.0 -1.0 1 1 500    // just black, others?`

Using OpenMP on the Cray (see Resources page on website)

<Demo, see class video>

Log into cray

- `ssh cray2.colostate.edu`
- `cd lustrefs`

Get the tar ball and unpack it

- `wget http://www.cs.colostate.edu/~cs560/Spring2012/CodeExamples/Mandel.tgz`
- `tar xzvf Mandel.tgz`

View the README file for compilation and execution directions

- `cc mandel.c mytimer.c ppm.c`
- `export OMP_NUM_THREADS=24`
- `aprun –d24 ./a.out`

View the output by copying file from cray to linux box

- `scp mandel.ppm carrot.cs.colostate.edu:/s/parsons/c/fac/mstrout/`
- `On CS machines: display mandel.ppm &`

OpenMP Constructs I

<Demo, showing constructs in mandel.c including gettimeofday()>

Header file

- #include <omp.h>
- Notice the #ifdef _OpenMP in the mandel.c example

Parallel region

- #pragma omp parallel { }
- Each thread runs a copy of the code.
- Unless specified private, variables are shared between threads.

For loop

- #pragma omp for
- If the following for loop is within a parallel region, then iterations of the loop are executed in parallel.
- #pragma omp parallel for // creates a parallel region for the for loop

OpenMP Constructs II

Reduction

- Functions/operators that are associative and commutative can be executed in any order.
- Example:
  - min(a,b,c) = min(a, min(b,c)) = min(min(a,b), c)
  - min(a,b,c) = min(c, min(b,a)) = min(b, min(c,a))

Reductions in OpenMP

- #pragma omp parallel for reduction(+:sum)
- Each thread gets a copy of of the reduction variable (e.g., sum) to execute that thread's set of iterations.
- The reduction operator is applied to all of the private reduction variables to get one result in the shared reduction variable.

Reduction Example (see the Resources page on website)

- <Demo reduction example on mac>
- <Draw the computation to illustrate the possible parallel schedules>

Parallel Performance Metrics

Speedup

$T_s(N)$ exec time for efficient serial computation on problem size N

$T(N,P)$ exectime for parallel version of computation on problem size N with P processors

speedup is the serial exec time divided by the parallel exec time

$$S = T_s(N)/T(N,P)$$

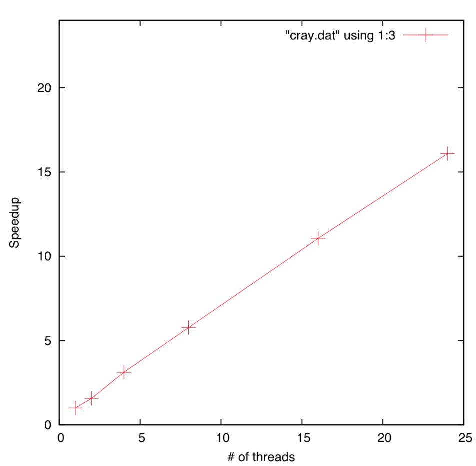
Efficiency

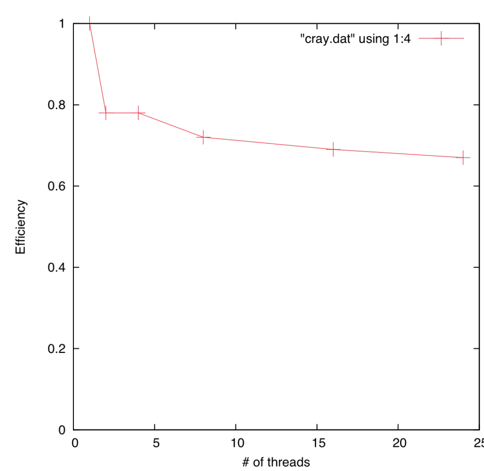
efficiency is the percentage of all the processing power that is being used

$$E = T_s(N)/(PT(N,P)) = S/P$$

Speedup and Efficiency of the Mandel Example

$$S = T_s(N)/T(N,P) \qquad E = T_s(N)/(PT(N,P)) = S/P$$





Amdahl's Law

How fast can we go?

- Assume that there is always some portion of the computation that is serial.
- The best we can do for speedup is

$$S = \frac{1}{(1-F) + F/S_F}$$

- Where F is the fraction of the computation that is parallel and S_F is the possible speedup for that fraction.
- Consequences: what if only 50% of the computation is parallelizable? 90%?

Scaling

Efficiency measures scaling

- 100% efficiency due to linear speedup is ideal, but not realistic.
- Strong scaling looks at efficiency as the problem size stays the same and the number of processors increases.

$$E = T_s(N)/(PT(N,P)) = S/P$$

- Weak scaling keeps the problem size per processor the same, but increases the overall workload as the number of processors increase.

$$E_W = T_s/T_p$$

Profiling Performance in Parallel Programs

Various tools

- HPCToolkit, TAU, CrayPat, …

Using TAU on the Cray

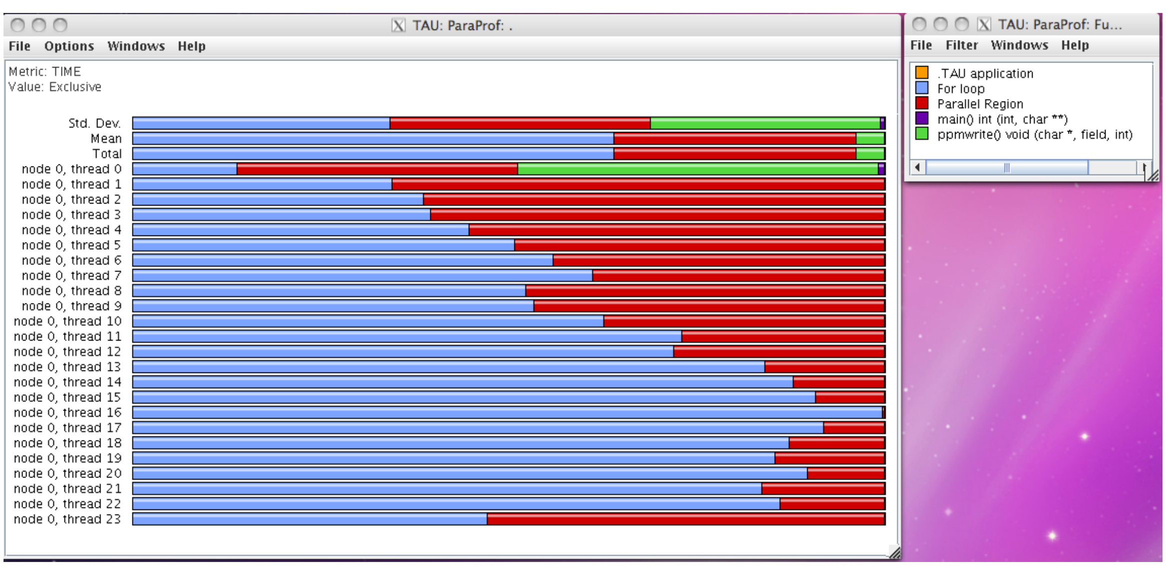
- Working on getting this installed on Cray and veges …
- Put following in source code

```
#include <Profile/Profiler.h>
TAU_PROFILE_TIMER(mt,"main()”,"int (int, char **)”,TAU_DEFAULT);
TAU_PROFILE_SET_NODE(0);
TAU_PROFILE_START(mt);
TAU_PROFILE_TIMER(pt, "Parallel Region", " " , TAU_DEFAULT);
TAU_PROFILE_START(pt);
…
TAU_PROFILE_STOP(pt);
TAU_PROFILE_STOP(mt);
```

Then execute, which will create profile files and use “paraprof .” to view

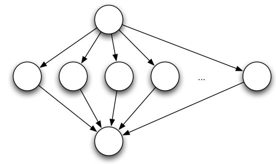
TAU Profile of mandel on the Cray

<Demo, go look at code again to understand variability>



Important Concepts

Parallel Computation as a DAG (directed acyclic graph)



Work

- The total amount of time for all of the tasks assuming we just add up the time for all the instructions per task.
- Let T_1 = work and T_P be the fastest parallel execution on P processors.
- The following bound holds:

$$T_P \geq T_1/P$$

Span, or Critical Path

- The longest path in terms of instructions in the DAG.
- Fastest parallel execution given infinite processors is span. T_∞
- Now we have another bound for the fastest parallel execution.

$$T_P \geq T_\infty$$

Load Balancing

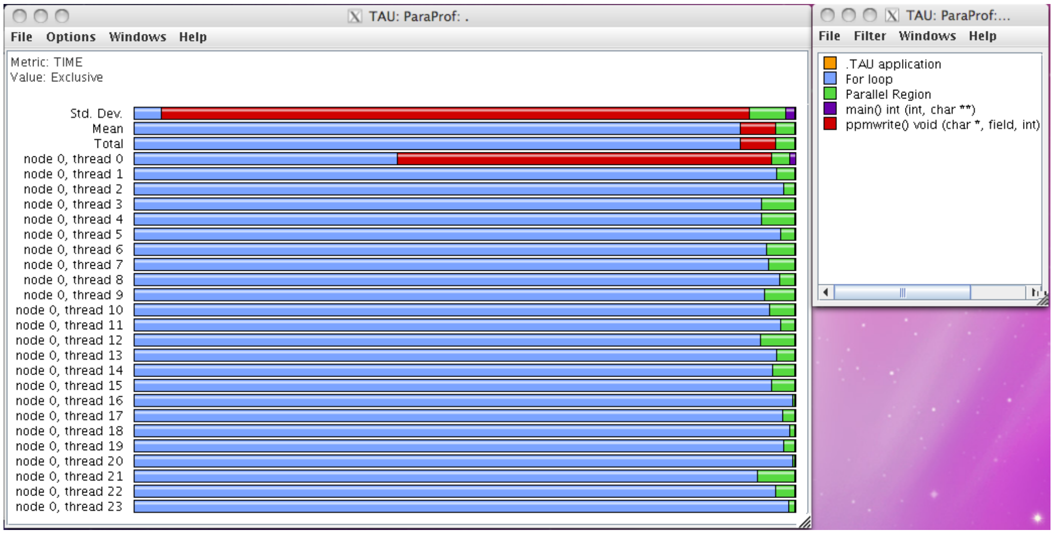
Problem

- Computing each pixel in the mandelbrot set can take a different number of iterations of the while loop.
- Default scheduling for OpenMP is implementation independent but probably evenly divides iterations between processors.

Possible solution, OpenMP scheduling clauses

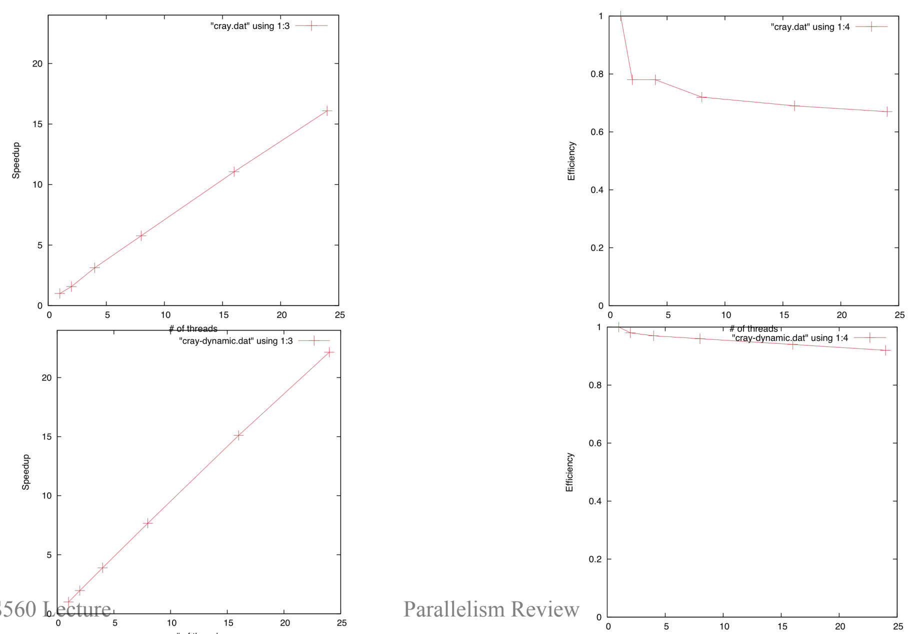
- #pragma omp for schedule(type [, chunk])
- type
  - **static**, iterations divided into pieces of size chunk and chunks are evenly divided among threads
  - **dynamic**, iterations divided into pieces of size chunk and dynamically scheduled on threads
  - … see tutorial for others

Performance Profile of mandel on Cray using DYNAMIC



Improvements to Speedup and Efficiency (Mandel)

$$S = T_s(N)/T(N,P) \qquad E = T_s(N)/(PT(N,P)) = S/P$$



Concepts

OpenMP

– Parallel regions
– Private variables
– For loops
– Reductions
– Scheduling the for loop

Performance Analysis for Parallelism

– Performance Profiling Tools: time command, gettimeofday(), Tau
– Speedup and efficiency
– Amdahl's law, isoefficiency, weak scaling, strong scaling
– Critical path, work, and span
– Load balancing

Next Time

Reading

– Roofline paper

Homework

– HW0 is due Wednesday

Lecture

– Complexity of Current and Future Computer Architectures

Terms (Definitely know these terms)

Parallelism terms

- Speedup and efficiency
- Amdahl's law
- Isoefficiency (will cover next week)
- Critical Path
- Work and Span

Performance terms

- MFLOPS – millions of floating point operations per second
- Load balancing

Some Thoughts on Grad School

Goals

- learn how to learn a subject in depth
- learn how to organize a project, execute it, and write about it

Iterate through the following:

- read the background material
- try some examples
- ask lots of questions
- repeat

You will have too much to do!

- learn to prioritize
- it is not possible to read ALL of the background material
- spend 2+ hours of dedicated time EACH day on each class/project
- have fun and learn a ton!

Isoefficiency