From Serial to Parallel

Intel® Parallel Studio
Background & Introduction

Stephen Blair-Chappell
Agenda

- Why Multicore?
- Why Parallelism?
- Overview of Intel Parallel Studio
Section 1
Intel Parallel Studio

Introduction to Intel Parallel Studio
Why is *everyone* going multi-core?

**Power Density Race**

- **Sun’s Surface**
- **Rocket Nozzle**
- **Nuclear Reactor**
- **Hot Plate**
- **Pentium® processors**

<table>
<thead>
<tr>
<th>Year</th>
<th>Power Density (W/cm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>'70</td>
<td>4004</td>
</tr>
<tr>
<td>'80</td>
<td>8008</td>
</tr>
<tr>
<td>'85</td>
<td>8086</td>
</tr>
<tr>
<td>'93</td>
<td>286</td>
</tr>
<tr>
<td>'00</td>
<td>386</td>
</tr>
<tr>
<td>'04</td>
<td>486</td>
</tr>
<tr>
<td>'08</td>
<td></td>
</tr>
<tr>
<td>'10</td>
<td></td>
</tr>
</tbody>
</table>
Multi-core: beating the **power\performance** barrier

- Over-clocked (+20%): 1.73x
- Design Frequency: 1.00x
- Under-clocked (-20%): 0.87x

**Relative single-core frequency and Vcc**
Industry’s First 45 nm High-K + Metal Gate Transistor Technology

- Improved Transistor Density: ~2x
- Improved Transistor Switching Speed: >20%
- Reduced Transistor Switching Power: ~30%
- Reduction in gate oxide leakage power: >10x

Enables New Features, Higher Performance, Greater Energy Efficiency
Cores, Cores & more Cores

• Today
  – Dual core
  – Quad core
  – Multi-socket solutions
• The Future
  – 6 & 8 cores
  – many-core
• R & D
  – 80 cores …

Performance comes from multi-core gains – but through parallelism
Four Steps in Moving to Parallel

Section 2
Steps in moving from Serial to Parallel

1. Architectural Analysis
2. Introducing Parallelism
3. Validating Correctness
4. Performance Tuning
5. Parallel
Key Questions - Design

- Is my program parallel?

- Where is the best place to parallelise my program?

- How can I get my program to run faster?

- What’s the expected speedup?
Key Questions - Code & Debug

• How?

• How difficult?

• Is my code still working?
Key Questions - Verify

- Is the parallelism correct?

- Do I have deadlocks or data races?

- Do I have memory errors?

- Does my program still work as intended?
Key Questions - Tune

• Do my tasks do equal amounts of work?

• Is my application scalable?

• Is the threading running efficiently?
OPEN BETA NOW: Intel® Parallel Studio
Intuitive development tools for multi core parallelism

For Microsoft Visual Studio* C++ architects, developers, and software innovators creating parallel Windows* applications.

Intel® Parallel Studio includes all three:
• Intel® Parallel Composer
• Intel® Parallel Inspector
• Intel® Parallel Amplifier

Note: Intel Parallel Advisor not yet available
From Serial to Parallel

Addressing the development lifecycle to harness parallelism

DESIGN
Gain insight on where parallelism will most benefit existing source code

CODE & DEBUG
Develop effective applications with a C/C++ compiler and comprehensive threaded libraries

VERIFY
Ensure application reliability with proactive parallel memory and threading error checking

TUNE
Enhance applications with easy-to-use performance analyzer and tuner
Parallel Studio vs Existing tools

- Integration
- End-to-end development
- Cost
- New features to support parallelism
Using Parallel Studio

Making Pi into a parallel program
Our running Example: The PI program

Numerical Integration

![Graph of F(x) = 4.0/(1+x^2) over the interval [0, 1].]

We know that:

$$\int_{0}^{1} \frac{4.0}{1+x^2} \, dx = \pi$$

We can approximate the integral as a sum of rectangles:

$$\sum_{i=0}^{N} F(x_i) \Delta x \approx \pi$$

Where each rectangle has width $\Delta x$ and height $F(x_i)$ at the middle of interval $i$. 
static long num_steps = 100000;
double step;
void main ()
{
    int i;
    double x, pi, sum = 0.0;
    step = 1.0/(double) num_steps;

    for (i=1;i<= num_steps; i++)
    {
        x = (i-0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
    }
    pi = step * sum;
}
Pi Program

double CalcSum(int i, double sum) {
    double x;
    x = (i + .5)*step;
    sum = sum + 4.0/(1. + x*x);
    return sum;
}

double CalcPi() {

    double pi, sum=0.0;
    for (int i=0; i<num_steps; i++) {
        sum = CalcSum(i, sum);
    }
    pi = sum*step;
    return pi;
}
Build Options

icl pi-serial.cpp /O2 /Zi /Qvec- /Ob0

- Optimise for speed
- Disable inlining
- Generate debug info
- Disable Vectorisation
## Serial Results

<table>
<thead>
<tr>
<th>test</th>
<th>/O2</th>
<th>/Ob0</th>
<th>/Qvec-</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS-1</td>
<td>√</td>
<td></td>
<td>n\a</td>
<td>4.34</td>
</tr>
<tr>
<td>MS-1</td>
<td>√</td>
<td>√</td>
<td>n\a</td>
<td>4.09</td>
</tr>
<tr>
<td>Intel-1</td>
<td>√</td>
<td></td>
<td>√</td>
<td>4.01</td>
</tr>
<tr>
<td>Intel-2</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>8.63</td>
</tr>
<tr>
<td>Intel-3</td>
<td>√</td>
<td></td>
<td></td>
<td>2.01</td>
</tr>
</tbody>
</table>
STEP 1
Finding the Hotspot
Key Questions - Design

• Is my program parallel?

• Where is the best place to parallelise my program?

• How can I get my program to run faster?

• What’s the expected speedup?
Design: Is my program parallel?

- **Advisor** (not yet available)

- **Intel® Concurrency Checker**
  - Free Tool
  - Use to measure concurrency
  - See Next Slide

- **Windows® Perfmon**
  - Part of Windows XP

- **Intel® VTune Analyzer**

- **Intel® Parallel Amplifier**
Intel Concurrency Checker


Avg CPU Utilization 90.22%
Elapsed Time 5000 ms
Parallel Time 0.00 %
Design: Where is the best place to parallelise my program?

• Get the biggest hotspot

• Determine the Calling Hierarchy

• Only parallelise where there is a decent amount of CPU Utilization

• Tools
  - Advisor
  - VTune
  - Code Coverage tools
Where to Parallelise - Amplifier

<table>
<thead>
<tr>
<th>Call Stack</th>
<th>Hotspot</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalcSum(int,double)</td>
<td></td>
</tr>
</tbody>
</table>

![Call Stack and Hotspot Diagram](https://via.placeholder.com/150)

Copyright © 2008, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners*
Design: How can I get my program to run faster?

- Check Interaction with the System
- Optimise using compiler flags
- Optimise application heuristics for best performance
- Analyse low-level stalls in CPU
Design: What’s the expected speedup?

• Use Amdhals Law

\[
\text{Speedup} = \frac{1}{[s+(1-s)/n + H(n)]}
\]

$s$ is serial part (fraction of 1)
$H$ is parallel overhead (ignore)
$n$ is number of cores

\[
s = \frac{4.186 - 3.634}{4.186} = 0.131
\]

\[
\text{Speedup} = \frac{1}{[0.131 + \frac{1 - 0.131}{2}]} = 1.77
\]

(i.e. new speed ~ 2.37 seconds)
STEP 2
Implementing Parallelism
Key Questions - Code & Debug

• How?

• How difficult?

• Is my code still working?
Code: How?

- OpenMP
- Language extensions
- TBB
- Threaded Libraries
- Native Threads
Pi Program

#include <omp.h>

double CalcPi()
{
    double pi, sum=0.0;

    #pragma omp parallel for
    for (int i=0; i<num_steps; i++)
    {
        sum = CalcSum(i,sum);
    }
    pi = sum*step;
    return pi;
}
Code: How difficult?

Easy as Pie

There’s a pun here somewhere ☺

….. Apparently
Results – Open MP

Amazing! Almost the speed we calculated (we expected ~ 2.37 seconds)
Code: Is my code still working?

Bother !!!!!!!!!!!!!!!!!!!!
Value of Pi is Wrong
From Serial to Parallel

STEP 3

Verifying the Parallel Code
Key Questions - Verify

- Is the parallelism correct?

- Do I have deadlocks or data races?

- Do I have memory errors?

- Does my program still work as intended?
Verify: Is the parallelism correct?

- Use threading tools to check for correctness
  - Intel Parallel Inspector
Verify: Do I have deadlocks or data races?

Problem: Data race at line 10
Pi Program

#include <omp.h>
double CalcPi()
{
    double pi, sum=0.0;

    #pragma omp parallel for
    for (int i=0; i<num_steps; i++)
    {
        #pragma omp critical
        sum = CalcSum(i,sum);
    }
    pi = sum*step;
    return pi;
}
Amazing!

Pi is Correct

Bother !!!!!!!!!!!!!!!!

It ran over 10 times slower
Pi Program

#include <omp.h>

double CalcPi()
{
    double pi, sum=0.0;

    #pragma omp parallel for reduction(+:sum)
    for (int i=0; i<num_steps; i++)
    {
        sum = CalcSum(i,sum);
    }
    pi = sum*step;
    return pi;
}
Verify: Does my program still work as intended?

Bingo!
Stategies to reduce time taken in Inspector

• Reduce depth

• Reduce Data set

```c
long long num_steps = 100000000;
if(argc == 2) {
    num_steps = atoi(argv[1]);
}
```
Data Races Fixed

No Problems Detected
The Inspector detected no problems at this analysis level. If this result is unexpected, try rerunning the target using a higher analysis level. Press F1 for more information.
Verify: Do I have memory errors?

No Memory Issues at deepest test

* Analysis level: Low
  * Call stack depth limit: 7

For best results, choose targets:
  * Compiled with debug information on, optimization off, and dynamic runtime library selected.
  * With small representative data sets
  (the overhead is proportional to the number of allocation calls)

Press F1 for Help
STEP

Tuning the Parallel application
Key Questions - Tune

• Do my tasks do equal amounts of work?

• Is my application scalable?

• Is the threading running efficiently?
Tune: Do my tasks do equal amounts of work?

- Yes – All green means a well-balanced threading
Lines of code

<table>
<thead>
<tr>
<th>Line</th>
<th>Source</th>
<th>CPU Time by Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>#include &quot;pi.h&quot;</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>double CalcSum(int i, double sum)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>{</td>
<td>0.459s</td>
</tr>
<tr>
<td>5</td>
<td>double x;</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>x = (i + .5)*step;</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>sum = sum + 4.0/(1.0 + x*x);</td>
<td>3.075s</td>
</tr>
<tr>
<td>8</td>
<td>return sum;</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>}</td>
<td>0.266s</td>
</tr>
</tbody>
</table>
Tune: Is my application scalable?

- Test it on quad \ many cores core etc
  - Use the env variable OMP_NUM_THREADS to help test
Tune: Is the threading running efficiently?

- Yes Waiting time is less than .042
- i.e. \( \frac{0.042}{2.671} \times 100 = 1.5\% \)