Performance on Android & OpenMP

RAMALINGAM R
(SAMSUNG INDIA)
Contents

1. Objective
2. Threads & Multicore
3. Parallel Programming
4. OpenMP
5. Conclusion
Objective

- Overview of SMP & Multicore architecture
- Parallel Programming
- OpenMP & Pthread concepts
Parallel Computing Background

- There are three ways to do anything faster:
  - Work Harder !!!!!
  - Work Smarter !!!!!
  - Get Help

- In Computing…
  - Work Harder => Increase the Processor Speed
  - Work Smarter => Use a better Algorithm

- Get Help => Use Parallel Processing !!!

- Parallel Processing:
  - Divide the problem among many processors
  - Each processor does a portion of the work assigned to it while interacting with its peers to exchange data
What is Parallelization?

- Parallelization is simply another **optimization technique** to get your results faster.
- **Parallelism refers to the concept of multiple threads executing simultaneously.**
- More than one processor is used to solve the problem.
- The "elapsed time" (wall clock time) comes down, but the total CPU time probably goes up.
- "Something" is Parallel if there is a certain level of independence in the order of operations:
  - A collection of program statements
  - An algorithm
  - A part of your program
  - The problem you're trying to solve
Concurrency, Parallelism, Shared Memory

- Concurrency - Two or more threads are in-progress at the same time

- Parallelism: Two or more threads are executing at the same time

- Shared Memory Model:
  - All threads have access to the same, globally shared, memory
  - Different threads are executing on different processors
  - Data can be shared or private
  - Shared data is accessible by all threads
  - Private data can only be accessed by the thread that owns it
  - Threads communicate via shared memory
  - Synchronization takes place, but it is mostly implicit
What is Thread?

- A Thread consists of a series of instructions with its own program counter ("PC") and state.
- A Parallel program executes threads in parallel.
- These threads are then scheduled onto processors.
- Threads need time sending data to each other and synchronizing ("communication").
Serial, Parallel Processing

• Example -1
Consider: 
A = B + C + D + E

Serial Processing:
A = B + C
A = A + D
A = A + E

Parallel Processing:
Thread 0: 
T1 = B + C
T1 = T1 + T2
Thread 1: 
T2 = D + E

• Example-2
for (i=0; i<8; i++)
a[i] = a[i+1] + b[i];

Thread 1
\[
\begin{align*}
a[0] &= a[1] + b[0] \\
\end{align*}
\]

Thread 2
\[
\begin{align*}
\end{align*}
\]

Every iteration in this loop is independent of the other iterations.
Single-Core and Multi-core Programming

- **SMP (Symmetric Multiprocessing or Symmetric Multiprocessor)** refers to an architecture in which **all processors access the same shared memory** and threads are executed on different processors by OS.
- For single-core chips, programmers write code so that they prioritize tasks in the order they must be performed to do the designated activity most efficiently and the OS then assigns the tasks to be run serially on the processor.
- Programmers write code for multi-core chips so that programmers will divide into tasks that the OS can assign to run in parallel on multiple cores.
- Multi-Core enables parallel multi-tasks/threads.

For single-core processors, the scheduler prioritizes tasks to be run serially on the chip (Single Core).

The Symmetric Multiprocessing Scheduler prioritizes tasks too, but it also must determine how best to run them in parallel on a Multicore chip’s processing units.
Multiprocessing is Here

- Multicore Chip systems are the future
- All chip manufactures instead of driving clock speeds higher, coming with Multicore architectures
- Multi-core processors embed two or more independent execution cores into a single processor package
- Intel has already developed a research chip with 80 cores.
- Some examples:

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Chip</th>
<th>#Cores</th>
<th>Arch</th>
<th>AMP</th>
<th>SMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>ARM11 MPCore</td>
<td>4</td>
<td>ARMv6</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Cavium</td>
<td>Octeon CN90</td>
<td>16</td>
<td>MIPS64</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Freescale</td>
<td>MPC8641D</td>
<td>2</td>
<td>PPC</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IBM</td>
<td>Cell</td>
<td>9</td>
<td>PPC64, DSP</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Raza</td>
<td>XLR 7-series</td>
<td>8</td>
<td>MIPS64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PA Semi</td>
<td>PA6T custom</td>
<td>8</td>
<td>PPC</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>TI</td>
<td>OMAP2</td>
<td>3</td>
<td>ARM, C55, IVA</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

![Diagram of Single Core vs. Multi-core](image)
Symmetric Multiprocessing (SMP) mode is the default to handle multiple processors under Linux.

- The OS must determine whether a processor is free and if it is, then a thread can be run on that processor even though the other processors may already be running other threads.
- Priority-based, preemptive scheduling uses multiple cores to run threads that are ready.
- The scheduler automatically runs threads on available cores.
- A thread that is ‘Ready’ can run on processor (n) if and only if it is of the same priority as the Thread(s) running on processor(n-1).
- In Linux, to maintain a balanced workload across CPUs, work can be redistributed, taking work from an overloaded CPU and giving it to an under loaded one.
- To make use of SMP with Linux on SMP-capable hardware, the kernel must be properly configured.
  - The CONFIG_SMP option must be enabled during Linux kernel configuration to make the kernel SMP aware.
- The 2.6 kernel maintains a pair of runqueues for each processor and each runqueue has tasks.
- When a thread needs to run (e.g. it is the highest priority runnable thread), and more than one core is available, the OS choose which core to use.
Parallel Programming Models

• Distributed Memory
  – MPI - Message Passing Interface (de-facto std)
• Shared Memory (Symmetric Multiprocessor)
  – POSIX Threads (standardized, low level)
  – **OpenMP (de-facto standard)**
    – Automatic Parallelization (compiler does it for you)
Rules for Parallel Programming for Multicore

• In Era of Multicore (Parallelism is essential)
• Think parallel. Understand where parallelism is, and organize your thinking to express it.
• **Code is about your problem, not about thread or core management.**
• Focus on writing code to express parallelism, but avoid writing code to manage threads or processor cores
• Leave the mapping of tasks to threads or processor cores to the compiler
• Do not use raw native threads (pthreads, Windows Threads, etc)
• Locks slows programs, reduce their scalability, so avoid Locks
OpenMP

- Compiler-assisted parallelization
- The *parallel* region is the basic parallel construct in OpenMP
- All the programmer needs to do is to tell OpenMP which code, loop, etc should be threaded.
- Programmers just need to add few OpenMP directives, Clauses for doing parallel ‘C’ code
- The OpenMP ‘C’ compiler determines how many threads to create and how best to manage them.
- OpenMP compiler and runtime library take care of these and many other details behind the scenes
- No need for programmers to add a lot of codes like POSIX Threads for creating, initializing, managing, and killing threads in order to exploit parallelism
- Thread executions are distributed among available processors
OpenMP motivation

- Vector addition
- Vector size: 1000 elements

**Conventional Programming:-**

```c
for (i=0; i<1000; i++)
    A[i] = B[i] + C[i];
```

**OpenMP Programming:-**

```c
#pragma omp for
for (i=0; i<1000; i++)
    A[i] = B[i] + C[i];
```

```
#pragma omp parallel
Thread 1    Thread 2    Thread 3    Thread 4

Join
```
Work sharing constructs

• `#pragma omp for`
  • Must be in the parallel region
  • Must precede loop
  • Splits loop iterations to threads
  • Can combine parallel and for directives
    `#pragma omp parallel for`
  • Omp do for do loops
    `#pragma omp do`
Data Environment

- Most variables are shared
- Stack variables in functions called from parallel region – private
- Automatic variables in a statement – private
- Loop indices – private with exceptions

- Directives for Data sharing
  - private (vars..) – variables are private
  - Shared (vars..) – variables are shared
  - default (shared|private|none) – change default behaviour
Pthreads - Fork & Join
### POSIX Pthreads Pros, Cons

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Multi-threaded programming in C with Pthreads is very flexible</td>
<td>• Complex code to develop for simple functionality to achieve</td>
</tr>
<tr>
<td>• You can have threads create threads recursively</td>
<td>• Number of lines of code to create threads are more…</td>
</tr>
<tr>
<td>• You can stop threads individually</td>
<td>• But in many cases, you don’t need that flexibility</td>
</tr>
<tr>
<td>• You can pretty much do anything you can think off</td>
<td>• Example:</td>
</tr>
<tr>
<td></td>
<td>– You want to compute the sum of an array</td>
</tr>
<tr>
<td></td>
<td>• Programmer must include header files, declare Pthreads data structures, and call Pthreads-specific functions for making multithread</td>
</tr>
<tr>
<td></td>
<td>• Platform dependent</td>
</tr>
<tr>
<td></td>
<td>• Considerable threading-specific code required</td>
</tr>
<tr>
<td></td>
<td><strong>Number of threads to use is hard-coded into the program.</strong></td>
</tr>
<tr>
<td></td>
<td>• It painful to deal with the Pthread API “just” to achieve this, over and over</td>
</tr>
<tr>
<td></td>
<td>– Ex: Pack the arguments into structures</td>
</tr>
</tbody>
</table>
Conclusion

• SMP architecture on Mobile is here to stay & will only increase in future
• Cores will get faster as it happened in the PC industry
• Parallel programming will be the main focus in times to come
• Pluses of OpenMP
  – Very easy to program
  – Can use for developing threaded applications
• Performance gain is function of many parameters
  – E.g. Memory bus sharing
Thank You
Questions ???