Intel® Software Tools and Application Development for Devices Powered by the Intel® Atom™ Processor
Objectives

At the end of this session, you will be able to:

- Overview Intel® Embedded Software Development Tools
- How to build and develop embedded software using Intel tools
- How to tune application performance using Intel tools and
- Use Intel® TBB for application development
Agenda

Intel® Embedded Software Development Tools Overview
Intel® Embedded System Software Development
Intel® Embedded System Software BLDK and System Debug
Application Performance Tuning
Threading for Performance with Intel® Threading Building Blocks
Intel® Tools Cover All These Device Categories

**Consumer electronic**
- Intel® Atom™ processor CE4100
- Intel® Media processor CE3100

**Mobile Internet Devices**
- Intel® Atom™ processor Zxx series

**Netbooks/Nettops**
- Intel® Atom™ processor Nxx series

**Embedded**
- Intel® Atom™ processor Zxx series

---

**Intel® Software Development Tools available**

- **Linux***
- **MeeGo/Linux***
- **MeeGo/Linux***
- **MeeGo/Linux***

**Intel® Software Development Products fully support Intel® Atom™ processors running MeeGo, Windows*, and RTOS**
## Intel® Software Development Tools Coverage

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Tools Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows*</td>
<td>- Intel® C++ Compiler for Windows*&lt;br&gt;- Intel® Integrated Performance Primitives Library (IPP)&lt;br&gt;- Intel® VTune™ Performance Analyzer&lt;br&gt;- Intel® Parallel Studio&lt;br&gt;- Intel® Threading Building Blocks</td>
</tr>
<tr>
<td>MeeGo/Linux*</td>
<td>- Intel® Embedded Software Development Tool Suite&lt;br&gt;- Intel® Application Software Development Tool Suite</td>
</tr>
<tr>
<td>RTOS</td>
<td>- Intel® C++ Compiler Professional Edition for QNX*&lt;br&gt;- Neutrino* RTOS</td>
</tr>
</tbody>
</table>

**“Application Suite“**
- For ISVs and Moblin Community – tune MeeGo applications for more performance and extend battery life of Intel® Atom™ processor powered devices

**“Embedded Suite“**
- For OEM/ODMs (+ their key ISVs) and OSVs – use a complete tools solution with a sophisticated JTAG debug solution for embedded system and application software design

Software Development Tools

MeeGo
Open Source Linux® SW Platform for Mobile & Embedded Devices
including Mobile Internet Devices (MID’s), Netbooks,
Automotive In-Vehicle Infotainment Systems

The MeeGo SDK
- Development guides, tutorials, sample code, API references
- Compliance Tools
- Project generator
- GNU Tools
- MeeGo Image Creator 2
- PowerTop

Intel® Software Development Tool Suite
- Intel® C++ Compiler
- Intel® Integrated Performance Primitives Library
- Intel® JTAG Debugger
- Intel® Application Debugger
- Intel® VTune™ Performance Analyzer

Intel® Tool Suites complement the open source MeeGo SDK
Intel® Software Development Tools

Intel® C++ Compiler
- Highly optimizing
- Full support for Intel® Atom™ processor
- GCC compatible

Intel® Integrated Performance Primitives Library
- Highly optimized multimedia functions
- Intel® Atom™ processor optimized

Intel® JTAG and Application Debuggers
- Intel® Atom™ processor and chipset support
- Kernel and low-level driver debugging
- Application debugging
- OS aware
- Built-in flash memory tool
- Execution trace support

Intel® VTune™ Analyzer
- Tune code actually running on device
- Performance bottleneck identification
- Tuning Assistant

Intel® Tools – a complete solution with more performance, and latest technology alignment

*Other names and brands may be claimed as the property of others
Agenda

Intel® Embedded Software Development Tool Suite Overview
Intel® Embedded System Software Development
Intel® Embedded System Software BLDK and System Debug
Application Performance Tuning
Threading for Performance with Intel® Threading Building Blocks
Intel® Tools for Embedded System Development

Cross Development
- Different host and target hardware
- Cross compile on host
- Download and debug with JTAG Debugger

Intel® C++ Compiler
- Build performance critical OS components and drivers
- Optimize for fast execution and fast OS switch into low power mode

Intel® JTAG Debugger
- Debug and identify issues in bootloader/firmware
- Debug and identify issues in OS kernel
- Debug and identify issues in device drivers
Using Intel® C++ Compiler for OS kernel development

• Use protected OS image build environment like MeeGo Image Creator 2

• OS kernels are highly optimized code. Recompile using different compiler – “hard work with limited benefit”

Typical approach:

• Install Intel® C++ Compiler into build environment

• Modify component makefiles to use ICC instead of GCC for parts that
  – Are multimedia or data volume, or data stream driven
  – Have a lot of direct interaction with user interface

• Improve overall OS responsiveness and end-user experience

Use Intel® C++ Compiler for spot optimizations
Building BLDK using ICC – Makefile Changes

Change compiler used from GCC to ICC in common.mak

Add Atom optimization –xSSE3_ATOM to CFLAGS
Agenda

Intel® Embedded Software Development Tool Suite Overview
Intel® Embedded System Software Development
Intel® Embedded System Software BLDK and System Debug
Application Performance Tuning
Threading for Performance with Intel® Threading Building Blocks
Build and Source Availability Requirements

ELF-Dwarf2 Linker output file - Makefile settings

- Use `-g` with compiler (gcc, icc) and `-debug` with linker (ld)
- Use `-Wall -g` with compiler if used as linker driver
- Define `DEBUG=1` in makefile

Currently not supported by BLDK IDE, Makefile modifications necessary. Update for external customer BLDK version planned
Building and debugging statically linked code

- Used for register testing, custom platform stress testing, hardware functionality testing and OS Bootloader

- For build
  - Use Intel® C++ Compiler or assembly
  - OS independent = separate build and link step.
    
    ```
    $ as test1.asm -o test1.o
    $ icc -c -O0 test2.c -o test2.o
    $ ld --image-base <address> --entry <address> --heap <size> -- stack <size> -o <image name> <input file list>
    $ objcopy -I elf32-i386 -O binary <image name in> <image name out>
    ```

- For debug
  - ensure consistent use of --g for all build steps
  - ensure link address and target download address are identical
Agenda

Intel® Embedded Software Development Tool Suite Overview
Intel® Embedded System Software Development
Intel® Embedded System Software BLDK and System Debug
Application Performance Tuning
Threading for Performance with Intel® Threading Building Blocks
Performance Optimization Principles

Re-compile
-xSSE3_ATOM (Atom switch / in-order scheduler)
IPO (interprocedural optimization)
PGO (program guided optimization)
OpenMP (works on multicore/HT only) - source modification

Implement library functions
Highly optimized multimedia/math library functions
OpenMP compiled (works on multicore/HT only)
Update application source code & build environment

Modify source code
Identify C and ASM - source spot optimization opportunities
Analyse results - update sources, rebuild, analyze again

Intel® Tools provide a complete spectrum of performance optimization Methodologies

Compiler: Intel® C++ Compiler
IPPC: Intel® Integrated Performance Primitives
VTune: Intel® VTune™ Performance Analyzer
<table>
<thead>
<tr>
<th>Compiler Features</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Significantly faster than GCC&lt;br&gt;High performing code maps directly into application quality and battery lifetime</td>
</tr>
<tr>
<td>In-order scheduler</td>
<td>Compiler optimization switch that re-arranges/optimizes application code to be executed with best performance on Intel’s Low-power Intel® Architecture technology&lt;br&gt;Better performance of system- and application software helps to reduce power consumption of a mobile device</td>
</tr>
<tr>
<td>Profile Guided Optimization</td>
<td>Multi-stage optimization method with feedback loop&lt;br&gt;Improves application performance by reducing instruction-cache thrashing, reorganizing code layout, shrinking code size, and reducing branch mispredictions</td>
</tr>
<tr>
<td>GCC Compatibility</td>
<td>Intel Compiler provides GCC language extensions and is source and binary code compatible with GCC&lt;br&gt;Saves efforts in porting/re-using existing code</td>
</tr>
</tbody>
</table>
• Intel® C++ Compiler 11.1
• Optimization Switch –xSSE3_ATOM
  – In order scheduler
  – IDIV → DIVB expansion
  – Arithmetic operations feeding addresses turned into LEAs
  – All stack adjusts done using LEAs

• Optimization Switch –aSSE3_ATOM
  – Code optimized for Intel® Atom™ Processor and a ‘generic’ x86 processor
  – Two code paths produced when necessary – min. a run time call to identify the processor used

Dedicated performance optimizations for the Intel® Atom™ Processor
Need For In-order Scheduler Support
- avoid dependency stalls

Consider code sequence:

\[
\begin{align*}
a &= b \times 7; \\
c &= d \times 7;
\end{align*}
\]

Representative assembly:

\[
\begin{align*}
1 & \text{ movl } b, \%eax \\
2 & \text{ imull } 7, \%eax \\
3 & \text{ movl } \%eax, a \\
4 & \text{ movl } d, \%edx \\
5 & \text{ imull } 7, \%edx \\
6 & \text{ movl } \%edx, c
\end{align*}
\]

- In some cases assembly code causes delays and dependency stalls which decrease the performance of application and performance critical code.
Need For In-order Scheduler Support
- avoid dependency stalls

Consider code sequence:

\[
\begin{align*}
    a &= b \times 7; \\
    c &= d \times 7;
\end{align*}
\]

-xSSE3_ATOM compiler switch

Processor cycles

Representative assembly:

\[
\begin{align*}
    1 \text{ movl } b, \%eax \\
    4 \text{ movl } d, \%edx \\
    2 \text{ imull } 7, \%eax \\
    5 \text{ imull } 7, \%edx \\
    3 \text{ movl } \%eax, a \\
    6 \text{ movl } \%edx, c
\end{align*}
\]

- Compiler switch –xSSE3_ATOM enables the in-order scheduler, which may improve application’s performance behavior

Model instruction pipeline and avoid dependency stalls by using the in-order-scheduler feature
Profile-guided Optimizations (PGO)

Use execution-time feedback to guide many other compiler optimizations

Helps I-cache, paging, branch-prediction

Enabled optimizations:
- Basic block ordering
- Better register allocation
- Better decision of functions to inline
- Function ordering
- Switch-statement optimization
- Better vectorization decisions
Highly optimized multimedia functions
- Images & video
- Communication & signal processing
- Data processing

Fully utilizing
- Intel® MMX™ technology
- SSE2, SSE3
- Multi-core / HT technology

Rapid application development

Cross-platform compatibility & code re-use

Outstanding performance

Use Intel® IPP libraries to concentrate on new features rather than optimizing application performance

Intel® IPP libraries to concentrate on new features rather than optimizing application performance
Get the best performance out of an application, by
Identifying optimization opportunities using the Intel® VTune™ Performance Analyzer

Questions to ask

Where do I spend most of my execution time?
Where do small optimizations have the biggest impact?
What hardware bottlenecks and dependency stalls can be easily avoided?
**Intel® VTune™ Performance Analyzer**

Identifies hard to find performance bottlenecks

**Features**

- Low overhead sampling
- No instrumentation required
- Monitor processor events like cache misses etc.
- View results in source or assembly

**Usage Model**

- Two components
  - Intel® VTune™ Performance Analyzer on host
  - Sampling Collector on the target
- Collect data on target and analyze it on the host
Sampling - How To Find Hotspots

Pick an event to sample and configure PMU
- Cache misses, branch mis-predictions, Dependency/pipeline stalls

Start SEP sampling routine and application
Performance Management Unit (PMU) periodically interrupts the processor
- Time-based
- Event-based: Triggered by the occurrence of a certain number of microarchitectural events

PMU

<table>
<thead>
<tr>
<th>Counter registers</th>
<th>SEP == ISR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event 1</td>
<td>&lt;0</td>
</tr>
<tr>
<td>Event 2</td>
<td>&lt;0</td>
</tr>
<tr>
<td>Event 3</td>
<td>&lt;0</td>
</tr>
<tr>
<td>Event 4</td>
<td>&lt;0</td>
</tr>
<tr>
<td>Event 5</td>
<td>&lt;0</td>
</tr>
</tbody>
</table>

General Purpose Event Registers
Dedicated Event Registers

- Numbers in counters define sampling rate (trade-off between sampling rate and impacting run-time behavior)

SEP == ISR

Collect
- Execution address in memory (CS:IP)
- OS process and thread ID
- Executable module loaded at that address
Write
- Information into *.TB5 file

- If you have symbols for the module, post-processing (in VTune) can identify the function or method at the memory address
- Line numbers from the symbol file can direct you to the relevant line of source code

Intel Confidential
Take Advantage of Sampling Data

The Intel® VTune™ Performance Analyzer tells you which module, function or routine could use some improvement.

Focus your application optimization efforts where it counts – Intel® VTune™ Performance Analyzer helps to analyze applications without source and binary instrumentation.
Agenda

- Intel® Embedded Software Development Tool Suite Overview
- Intel® Embedded System Software Development
- Intel® Embedded System Software BLDK and System Debug
- Application Performance Tuning
- Threading for Performance with Intel® Threading Building Blocks
You specify **tasks** (what can run concurrently) instead of threads
- Library maps your logical tasks onto physical threads, efficiently using cache and balancing load
- Full support for *nested parallelism*

**Targets threading for scalable performance**
- Portable across Linux*, Mac OS*, Windows*, and Solaris*

**Emphasizes scalable data parallel programming**
- Loop parallelism tasks are more scalable than a fixed number of separate tasks

**Compatible with other threading packages**
- Can be used in concert with native threads and OpenMP

Open source and licensed versions available
Components of TBB (version 2.1)

Parallel algorithms
- parallel_for (improved)
- parallel_reduce (improved)
- parallel_do (new)
- pipeline (improved)
- parallel_sort
- parallel_scan

Concurrent containers
- concurrent_hash_map *
- concurrent_queue
- concurrent_vector (all improved)

Task scheduler *
With new functionality

Synchronization primitives
- atomic operations
- various flavors of mutexes (improved)

Utilities
- tick_count
- tbb_thread (new)

Memory allocators *
- tbb_allocator (new), cache_aligned_allocator, scalable_allocator
Tasks are light-weight entities at user-level
- Intel® TBB parallel algorithms map tasks onto threads automatically
- Task scheduler manages the thread pool
  Scheduler is unfair to favor tasks that have been most recent in the cache
- Oversubscription and undersubscription of core resources is prevented by task-stealing technique of TBB scheduler
Best known example is C++ STL
Enables distribution of broadly-useful high-quality algorithms and data structures
Write best possible algorithm with fewest constraints
- Do not force particular data structure on user
- Classic example: STL std::sort
Instantiate algorithm to specific situation
- C++ template instantiation, partial specialization, and inlining make resulting code efficient

Standard Template Library, overall, is not thread-safe
Intel® Threading Building Blocks is a parallel programming model for C++ applications

- Used for computationally intense code
- A focus on data parallel programming
- Uses generic programming

- Intel® Threading Building Blocks provides
  - Generic parallel algorithms
  - Highly concurrent containers
  - Low-level synchronization primitives
  - A task scheduler that can be used directly

- Know when to select Intel® Threading Building Blocks, the OpenMP API or Explicit Threading
Summary

- Intel Software Development **Tool for OEMs, OSVs**, ("Embedded Suite") and **ISVs** ("Application Suite") **cover the entire cycle of SW development**

- **Intel® Tool Suites** for Intel® Atom™ Processors **complement** the open source MeeGo SDK

- Intel Tool Suites provide a **complete spectrum of performance optimization** methodologies (compiler switches, IPP multimedia libs, performance bottleneck analysis with VTune)

- **Intel® C++ Compiler for spot optimizations in System Software**, e.g. performance critical drivers, codecs, and **applications in general**
  - Use Intel® C++ Compiler for higher performance of your sensitive device drivers and codecs

- **Intel JTAG debugger** for in-depth system software debugging with full **Si/SoC/chipset awareness**
  - Find OS kernel and driver issues faster with Intel's GUI driven system-level JTAG debugger

OS bring-up, Hardware test code deployment, driver development. The Intel® C++ Software Development Tool Suite has you covered.
Product Page:
http://www.intel.com/software/products/atomtools

Knowledge Base:

Forum:
Q&A