Exploring Multi-core Processors using Realistic Signal- and Image-processing Application Benchmarks

Ray E. Artz, Brian J. Loe, Janet Pavelich

Lockheed Martin MS2 Tactical Systems

3333 Pilot Knob Road, Eagan, MN 55121

{ray.e.artz, brian.j.loe, janet.pavelich}@lmco.com

Jules Bergmann

CodeSourcery

9978 Granite Point Court, Granite Bay, CA 95746

jules@codesourcery.com

High Performance Embedded Computing (HPEC) Workshop

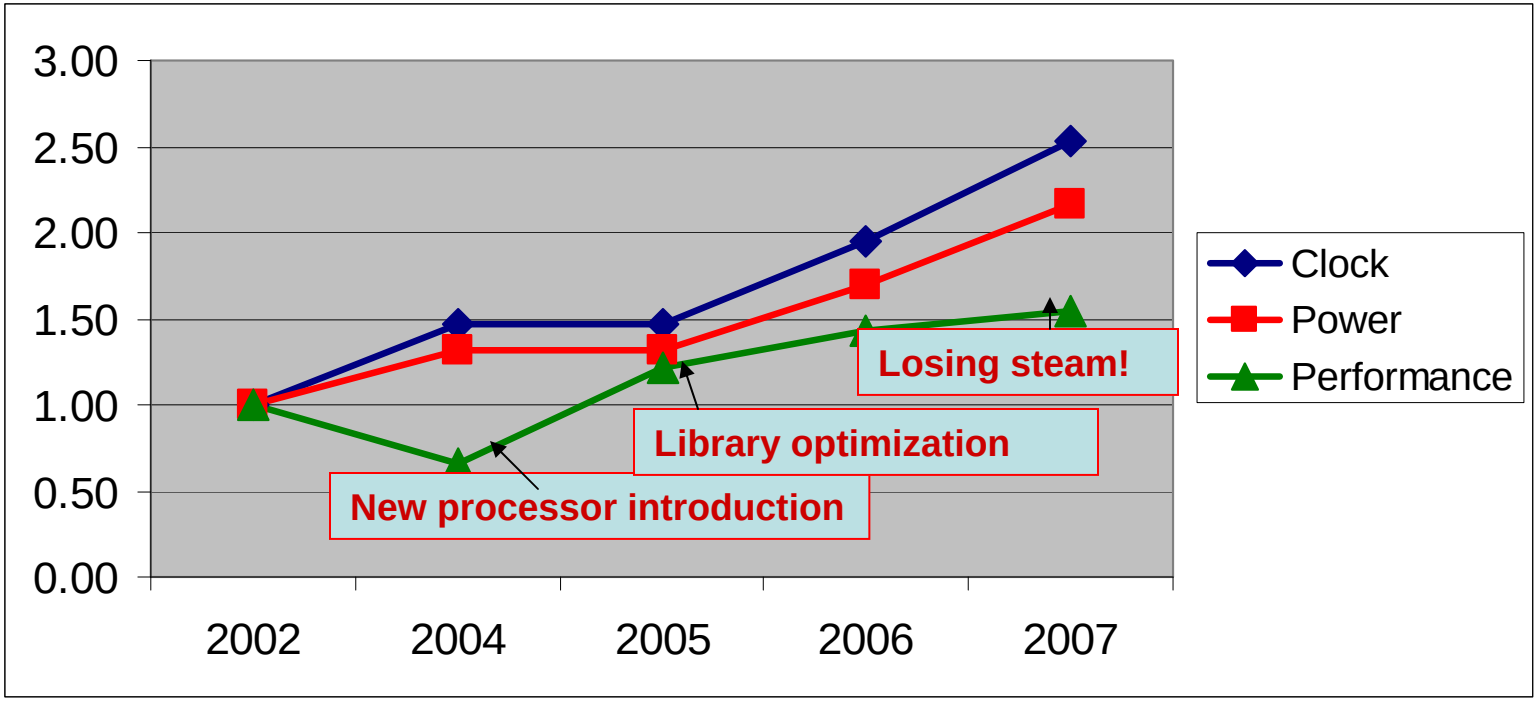
18-20 September 2007

Presentation outline

- Overview
  - Motivation, purpose, objectives, scope
- P.A. Semi PA6T-1682M PWRficient™ Processor
- Benchmarking laboratory environment
  - Target, host, and software-development platforms
  - Software development environments and vector libraries
- Project activities to date (June-August 2007)
- Benchmarks – descriptions and results
  - Performance and power
- Multi-core operation
- Software development practices and experiences
- Wrap-up
  - Looking forward

Overview: Motivation

Multi-core, low-power: Why do we care?



- **Typical DoD program experience in the past five years:**
  - **Over multiple processor generations, processor clock went up 2.5x**
  - **Processor performance went up 1.5x, but power went up 2.2x**
  - **Unless the pattern changes, future performance growth will be limited by power and cooling considerations**

Overview: Project purpose

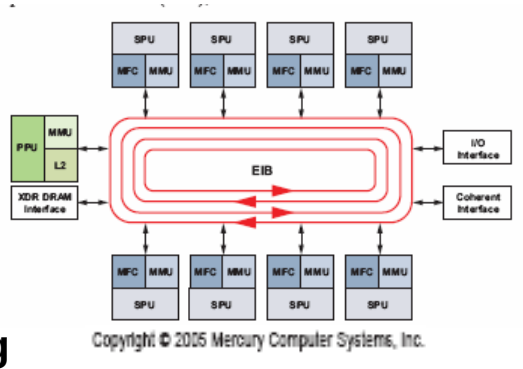
- **Primary**
  - **To gain knowledge of emerging multi-core processors with possible application in DoD high-performance embedded-computing platforms. Processor issues include:**
    - **Performance**
    - **Power consumption**
    - **Multi-core operation**
- **Secondary**
  - **To gain experience with various single- and multi-core software libraries and software-development environments for possible use in DoD programs. Software issues include:**
    - **Portability**
    - **Productivity**
    - **Performance**
    - **Multi-core operation**

Overview: Project objectives and scope

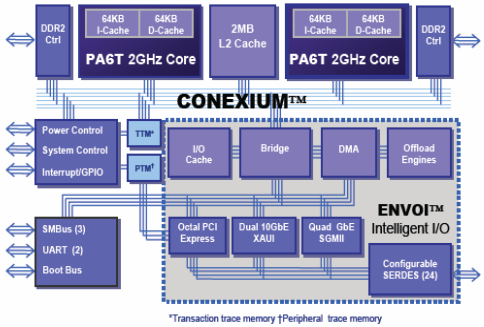
- **Near-term (results reported here)**
  - **Use several realistic application-level benchmarks to study the performance of:**
    - **PWRficient™ PA6T-1682M dual-core processor**
  - **Also, report on software-development practices and experiences related to building benchmarking software using:**
    - **Mercury SAL and CodeSourcery VSIPL++**
- **Longer-term**
  - **Use similar benchmarks to study:**
    - **Full-production versions of the PA6T-1682M family**
    - **IBM Cell Broadband Engine™ (BE)**
    - **Use of parallel vector libraries such as Mercury Multi-core Framework (MCF) and Parallel VSIPL++**

Overview: Scope
Why these two multi-core processors?

- **IBM Cell Broadband Engine™ (BE) processor offers very high performance but also presents significant engineering challenges**
  - **Demonstrates excellent performance per watt, especially on kernel benchmarks, but**
  - **Is a very-high-power component (100 watts?)**
  - **Portable and productive software development for the Cell BE is a challenge, though tools are emerging**
- **Semi PA6T-1682M processor also offers high performance, but without so many challenges**
  - **Peak performance per component is far lower than that of the Cell BE, but**
  - **Power consumption per component is much lower**
    - **Easier to cool via conventional means**
  - **Power Architecture™ (including AltiVec™) instruction set and conventional programming models permit easy and direct ports of existing vector libraries and application software**



Used with permission from Mercury Computer Systems for HPEC 2007



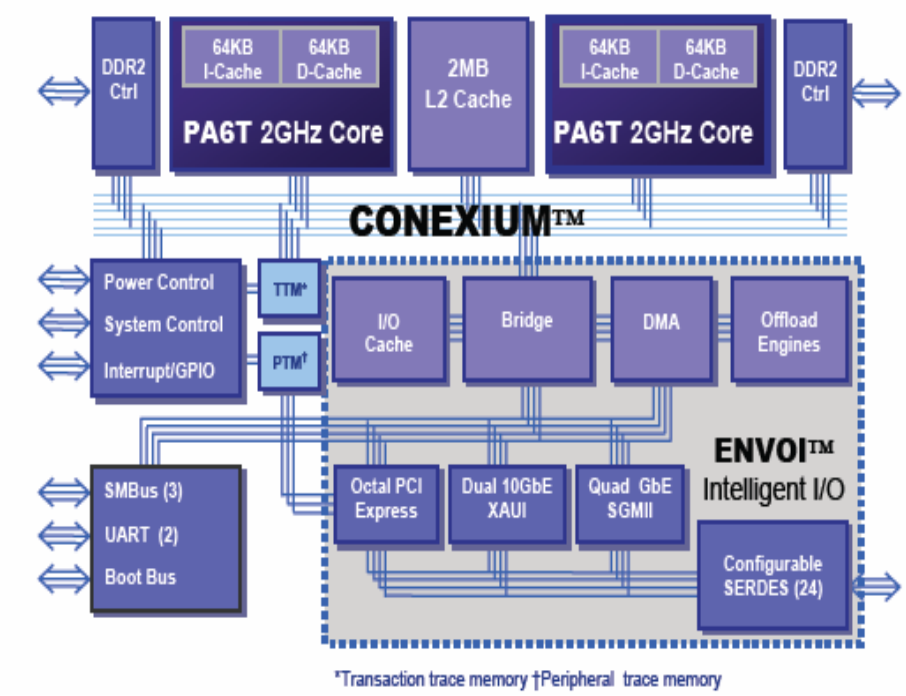
Used with permission from P.A. Semi for HPEC 2007

P.A. Semi PWRficient™ PA6T-1682M

P.A. Semi, Inc.

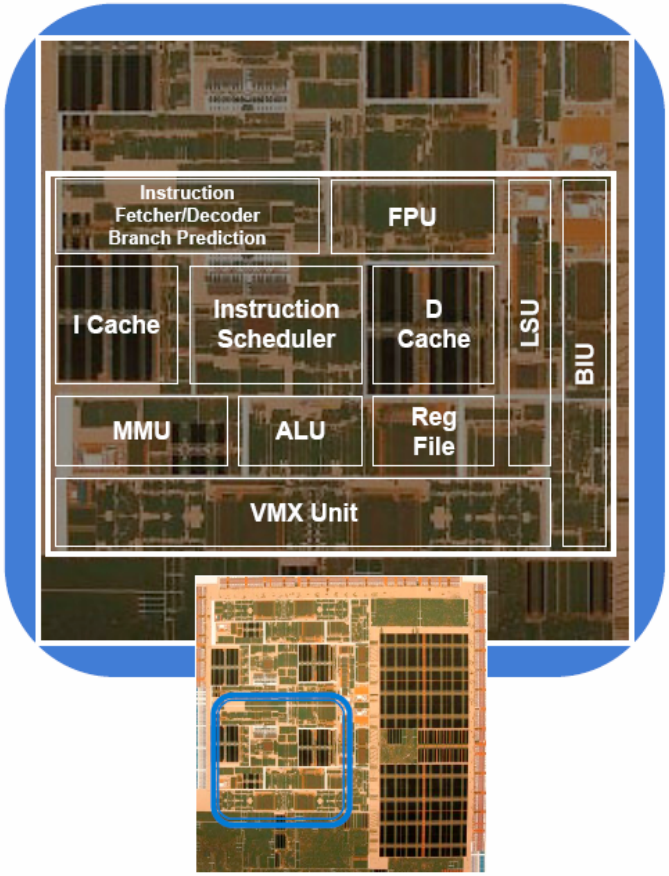
- Fabless processor company based in Santa Clara, CA
- Power Architecture™ Licensee
- Designer and producer of low-power multi-core processors
- Began shipping sample quantities and evaluation kits 4Q 2006
- Producing dual-core PA6T 1682M as first of a family of multi-core processors
- Per public press releases, committed to supplying PA6T components for inclusion in products by:
  - Mercury Computer Systems
  - Curtis Wright
  - Extreme Engineering Solutions
  - Themis
  - Others

PA6T-1682M Block diagram



Used with permission from P.A. Semi for HPEC 2007

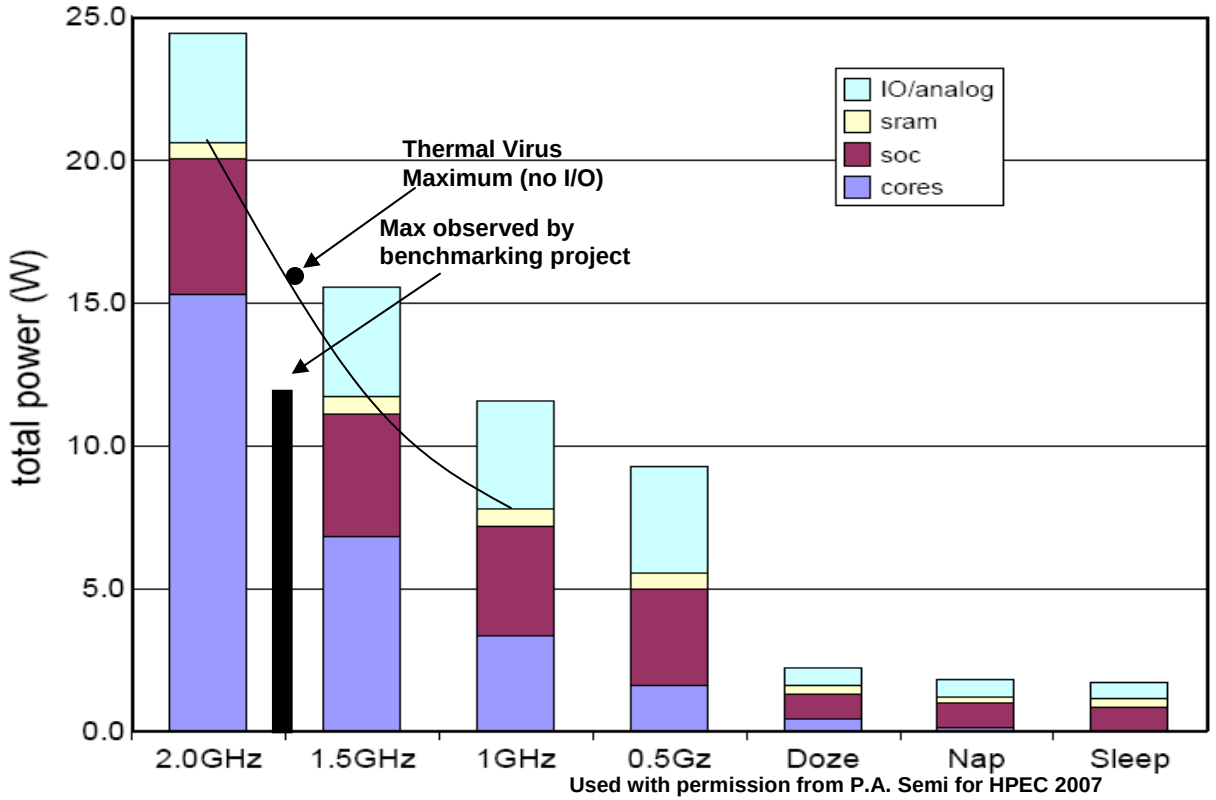
P.A. Semi PWRficient™ PA6T Core



- **Functionality**
  - 2.0GHz 64-bit core with FP and VMX
  - Super-scalar, out-of-order design
    - Quad-fetch, triple issue
  - 64KB L1 I + D Caches
  - Interface to on-chip coherent bus
  - Idle and sleep modes
- **Technology**
  - 65nm CMOS
  - 0.6–1.2V V_{dd}
  - 0.3V V_t
  - 11M logic transistors
- **Power**
  - 7W @ 2.0GHz
- **Performance**
  - SPECint®2000 >1000 per core
  - SPECfp®2000 >2000 per core

P.A. Semi PWRficient™ PA6T-1682M

Measured maximum power at 100 deg C



Used with permission from P.A. Semi for HPEC 2007

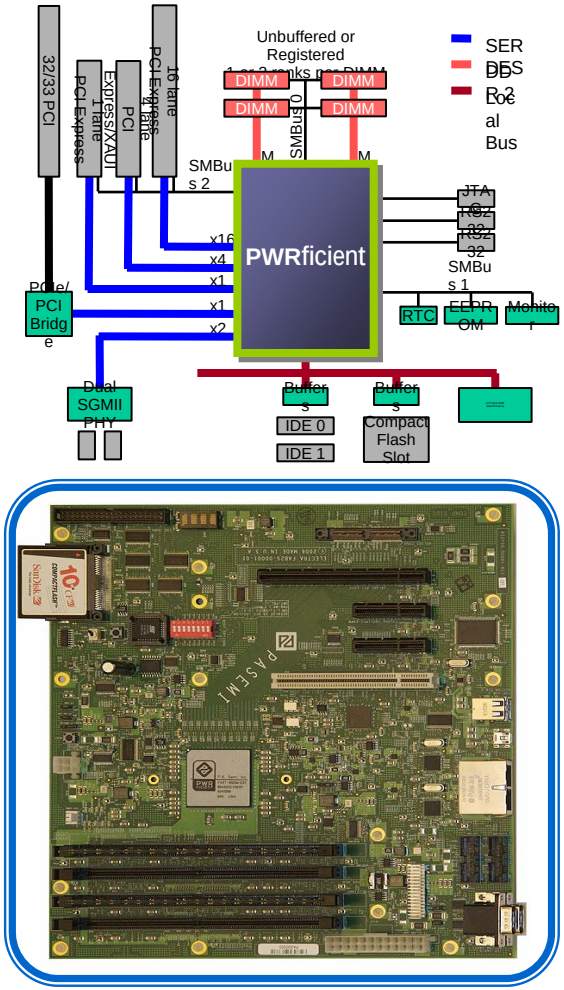
- Chart shows maximum-power measurements made by P.A. Semi using artificial "thermal-virus" software designed specifically to make maximum simultaneous use of all circuitry
- Maximum power utilization observed in this benchmarking study was 11.9 watts total chip power with A.2 silicon, simultaneous 2-core operation (but no I/O) at 1.7 GHz

Benchmarking laboratory environment
Target, host, and code-development platforms

- **Target platform: P.A. Semi Electra evaluation kit (see next slide)**
- **Host platform: IA86/Linux Desktop computer networked to Target**
  - **Sourcery G++ from CodeSourcery**
    - **IDE for C++, based on Eclipse IDE and GNU toolchain**
    - **Cross-compiler for Power/Linux target, based on GNU G++**
  - **Cross-compiled libraries for Power/Linux target**
    - **Mercury SAL (pre-release version)**
    - **CodeSourcery VSIPL++ 1.3**
- **Development platforms: Dell Laptop computers w/ Windows NT**
  - **Principal platforms for source code development and debug**
  - **Sourcery G++ from CodeSourcery**
    - **IDE for C++, based on Eclipse IDE and GNU toolchain**
    - **Cross-compiler for Power/EABI target, based on GNU G++**
    - **EABI Power-target simulator**
  - **Cross-compiled libraries for Power/EABI target (simulated)**
    - **Mercury CSAL**
    - **CodeSourcery VSIPL++ 1.3**

Benchmarking laboratory environment
Target platform

- **P.A. Semi RDK Electra Board**
  - ATX form factor in standard PC chassis
  - PA6T-1682M Processor
    - A.2 silicon, 1.7 GHz
  - Four DIMM sockets, two populated
    - w/ 512MB DDR-2 DIMM's
  - IDE port on board, drive installed by project team
- **RDK Electra software**
  - Common Firmware Environment (CFE) boot monitor
  - Board Support Package for Linux
  - Linux Kernel



Used with permission P.A. Semi for HPEC 2007

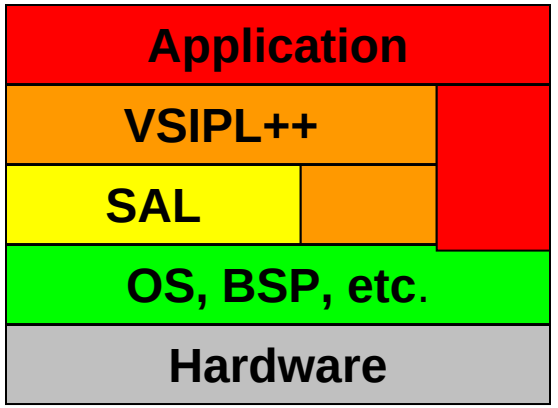
Benchmarking laboratory environment
SAL and VSIPL++ vector libraries

- **Mercury Scientific Application Library (SAL)**
  - **Mature C-language vector library**
  - **Over 10 years sustained development**
  - **Well-deserved reputation: “gold standard” for performance**
  - **Project used a pre-release (not-yet-optimized) version of SAL for Power/Linux supplied by Mercury Computer Company**
- **Vector-, Signal-, and Image-processing Library (VSIPL++)**
  - **Open standard, modern object-oriented C++ API**
  - **Development sponsored by DoD**
  - **Sourcery VSIPL++ by CodeSourcery is an optimized implementation of the standard**
    - **Designed for portability, productivity, and performance**
    - **Employs multiple means to achieve performance**
      - **Including linking to optimized libraries (e.g., SAL)**
  - **Version of VSIPL++ used for benchmarking software:**
    - **CodeSourcery VSIPL++ 1.3 compiled for generic Power/LINUX architectures**

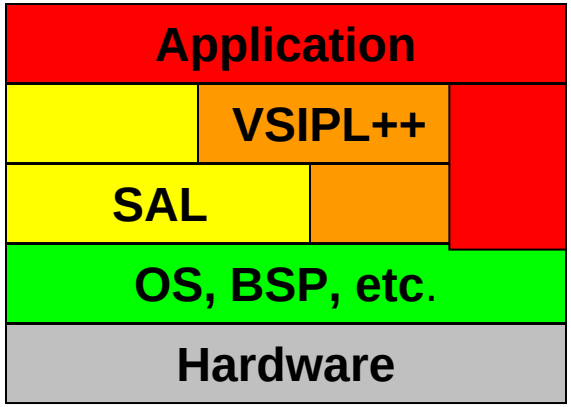
Benchmarking laboratory environment

SAL and VSIPL++ used in two configurations

1. VSIPL++ API with “implicit” SAL
   - SAL used by VSIPL++
   - SAL not directly visible to the application
   - Maximizes software portability



2. Both VSIPL++ and SAL API’s visible to the application
   - VSIPL++ used at the top-level for object-oriented design
   - SAL used explicitly for selected computations
   - Gives developer extra opportunities to directly optimize performance



Project activities
Busy June!

- Electra system arrived and hardware worked immediately
- Project had typical start-up with some “growing pains”
  - Network installation, corporate internet firewalls, LINUX issues
  - Project team’s first use of C++ and VSIPL++
- Then “With a Little (actually, lots of!) Help from Our Friends”:
  - PA-Semi sends us a free hard-drive with pre-installed software
  - We evaluate, then purchase, CodeSourcery G++ IDE for both Host and Development Platforms: this greatly facilitates SW development
  - Mercury gives us permission to use pre-release SAL libraries on our PA-Semi target, plus Mercury CSAL for desktop use
  - CodeSourcery contributes VSIPL++ support – to the extent that Jules Bergmann joins our team as a co-author

Project activities
Busier July !!

- Stabilized benchmarking software environments
  - Installed Sourcery G++ EABI simulator on development platforms for source-code operation, validation, and debug
  - Installed and integrated VSIPL++ and SAL libraries on all three platforms: development, host, and target
    - CSAL was used on development host
  - Established near-seamless transition from development platform to target platforms
- Developed most of the benchmarking code
  - Electro-optical benchmarking code almost completed
  - Acoustic Beamformer benchmarking code started
- Started data collection

Project activities
Hazy, Crazy, but-not-Lazy Days of August!!!

- Added additional code instrumentation and collected data
- Checked data for anomalies, re-collected as necessary
- Tuned and optimized (to the extent possible in 3 weeks)
  - Application code and G++ compiler options (Thanks, Mercury and CodeSourcery!)
  - VSIPL++ library (Thanks, CodeSourcery!)
  - New PA6T Linux Kernel and PA6T run-time settings (Thanks, P.A. Semi!)
- Regretted having so little time with so many untapped opportunities to improve performance further
  - BUT: knowing it leaves us lots to talk about at HPEC 2008
- Completed HPEC presentation material in time to clear corporate-release process and still meet 31 Aug deadline
- WHEW! But it's great to be here!

Benchmarks
Kernel versus application-level

- Kernel benchmarks
  - Give detailed information about how individual algorithms perform in isolation
    - FFT's, matrix multiplies, etc.
  - Are widely reported in technical and marketing literature
  - Are very valuable for performance prediction, but are easily misused
  - Are often overly optimistic compared to actual performance
    - Likely to reflect performance on the most highly-tuned, parallel code in a signal- or imaging-processing program
- Application-level benchmarks
  - Can be too application-specific
    - Extrapolation from one application to another can be problematical
  - Nonetheless provide a valuable "sanity check" on kernel-benchmark results
- This study is focused on application-level benchmarks
  - Some kernel results will be collected as time and resources permit

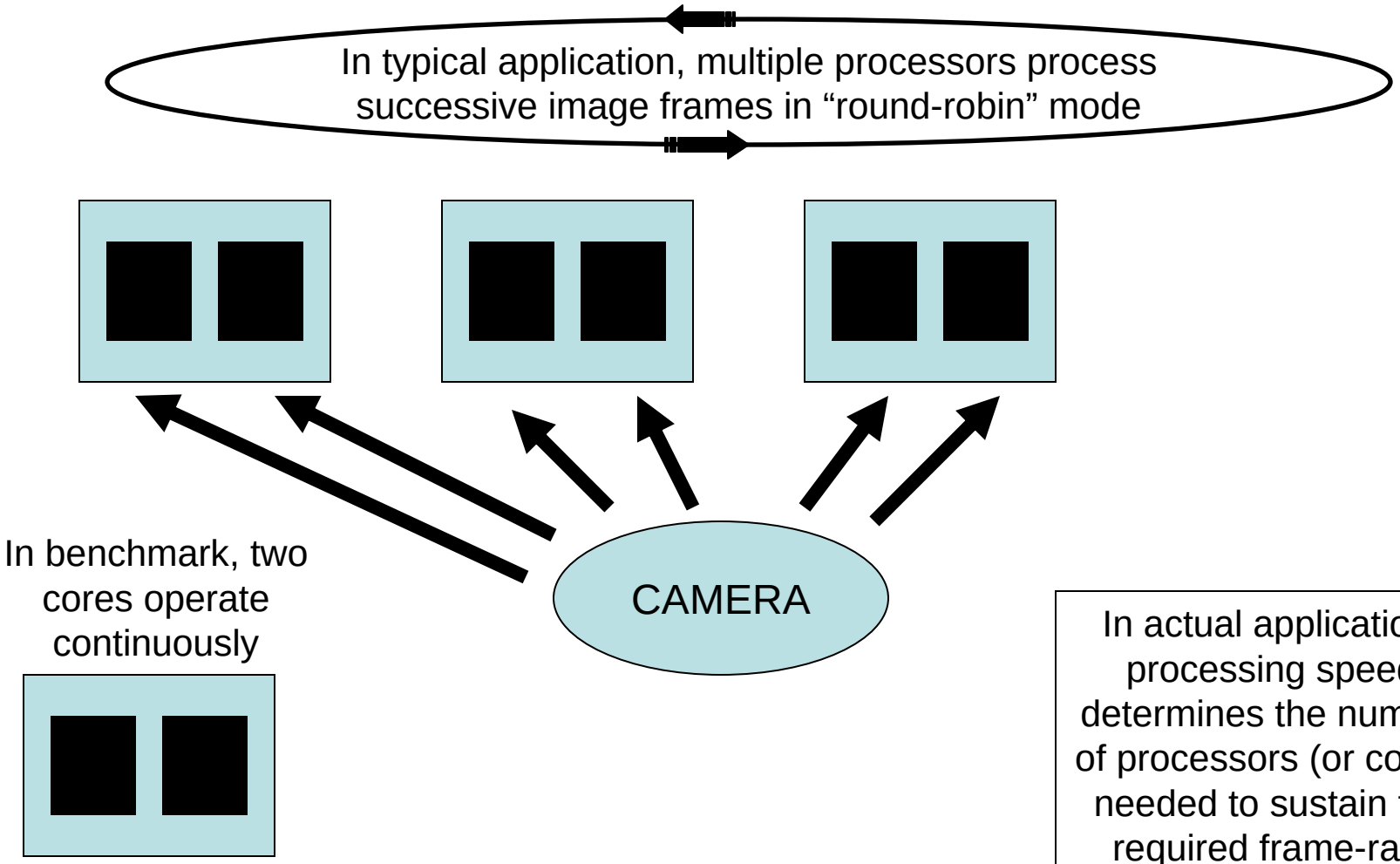
Benchmarks
Electro-optical (EO) application

- **Electro-optical benchmarking software is similar to code used in actual DoD applications, but:**
  - **Is organized for benchmarking convenience and data collection**
  - **Uses VSIPL++ API at the top level for object-oriented design**
  - **Can be run in either of two vector-library configurations (see Slide 13):**
    1. **VSIPL++ API: Only VSIPL++ API visible to application**
    2. **VSIPL++/SAL API: Both SAL and VSIPL++ API's visible to application**

    In either configuration, VSIPL++ is permitted to link to SAL automatically for performance

NOTE: Because the benchmarking code closely resembles actual application code, and is based on Lockheed Martin Proprietary algorithms, this presentation will present only aggregate performance data for it, with no detailed breakdown or algorithmic detail.

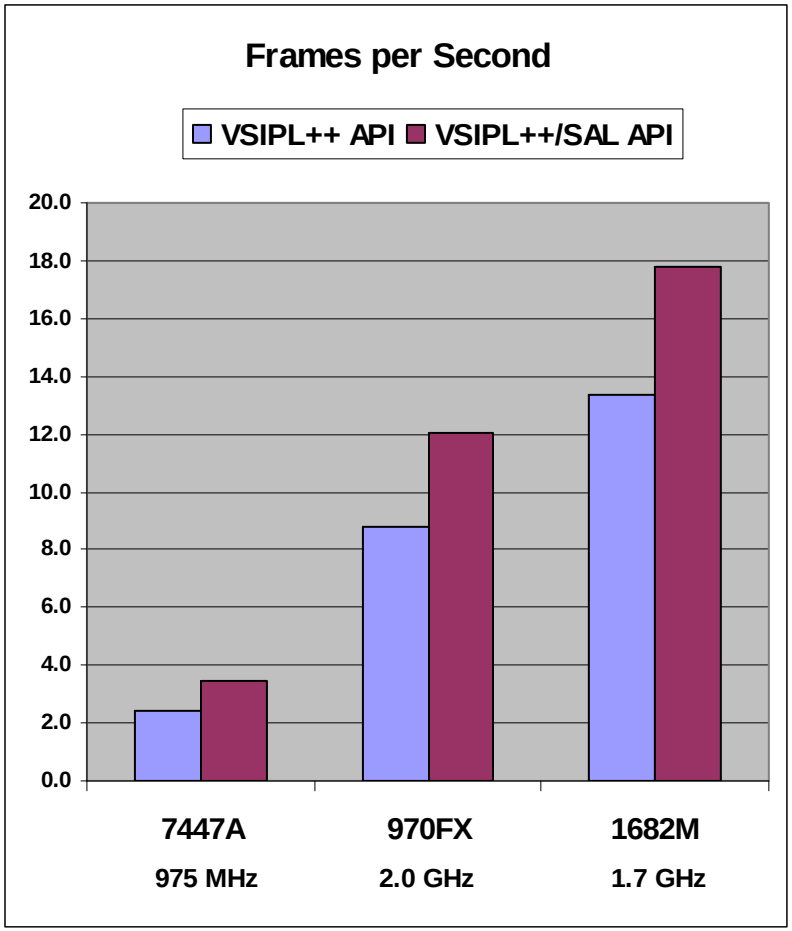
Benchmarks
Electro-optical (EO) application



In actual application, processing speed determines the number of processors (or cores) needed to sustain the required frame-rate

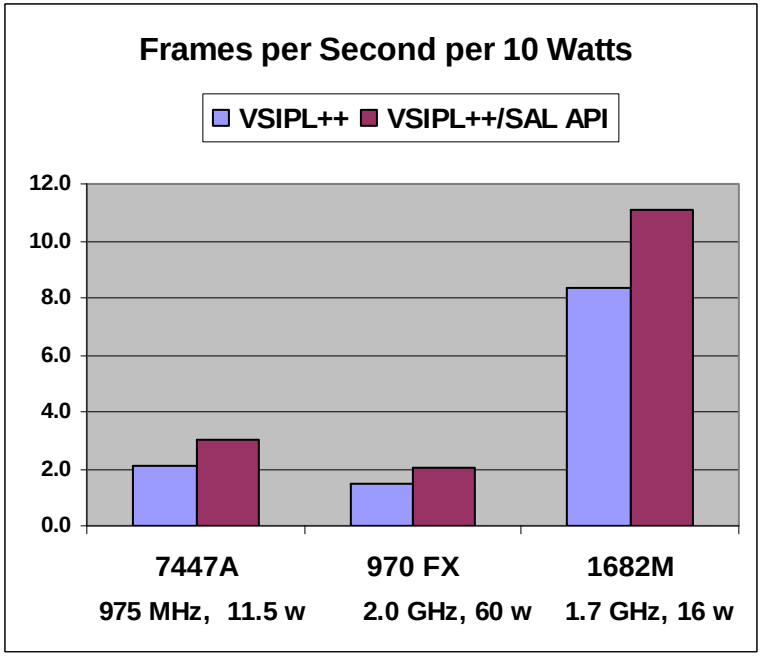
Benchmarks
Electro-optical performance

Raw performance



Performance scaled to power

Based on "worst-case" power (excluding IO power) per vendor data sheets



Note: "Frames per second" as used here is based on an arbitrary benchmark and does not correspond to actual frame processing time in any real system

Benchmarks
Acoustic beamformer application

- Benchmarked in VSIPL++ API mode only
- SAL not visible to application
- VSIPL++ linked to SAL for performance

Preprocessing steps

Data Read
FIR filter
Decimate

Calculate
Beam Directions
Steering Vectors

Outer loop (Frames in time domain)

Linear De-trend → Blackman Shading → FFT

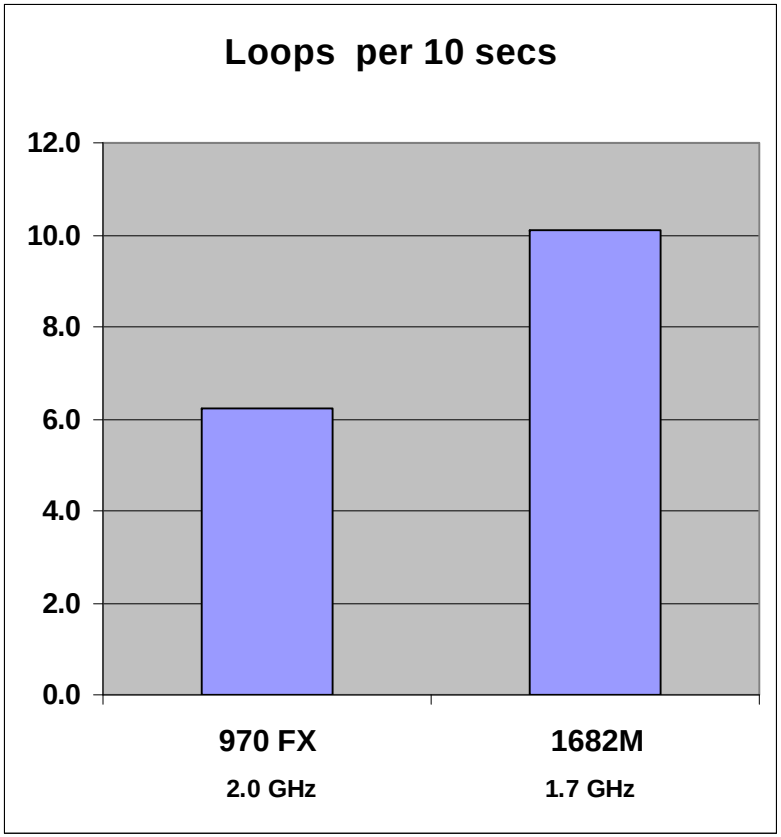
Inner loop (Frequencies)

Calculate Weights → Calculate Beams

16 Channels
Sample Rate = 44.1 kHz
Low Pass Filter
Decimate to 4.41 kHz

Benchmarks
Acoustic beamformer performance

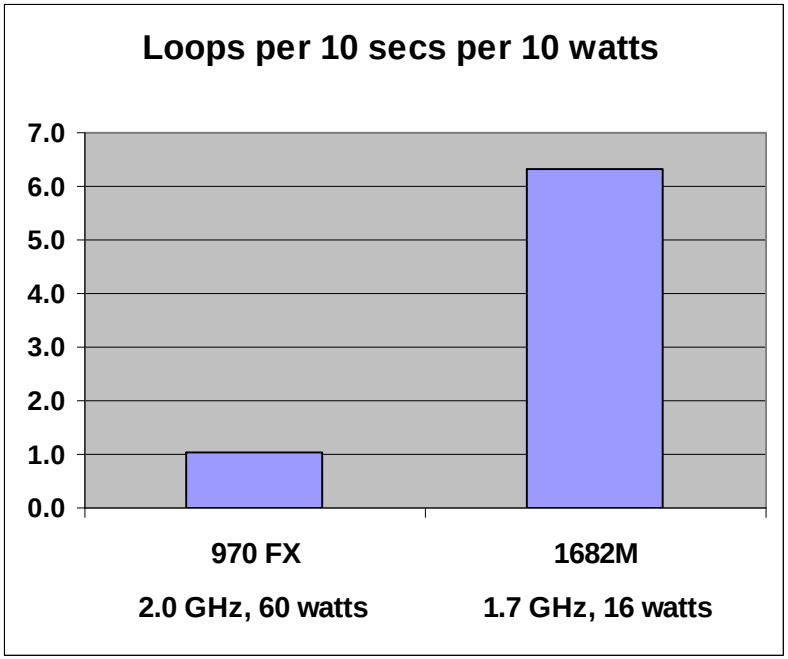
Raw performance



Note: 7447A benchmark results not available in time for HPEC 2007 presentation

Performance scaled to power

Based on "worst-case" power (excluding IO power) per vendor data sheets



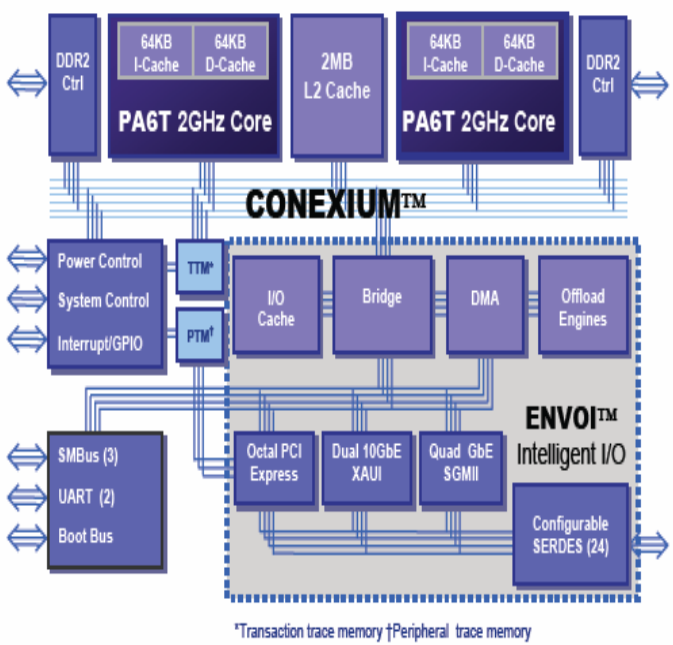
Note: "Loops per second" as used here is based on an arbitrary benchmark and does not correspond to cyclic processing time in any real system

Multi-core operation
Dual-core performance

Q: How does dual-core operation compare to one-core operation on the PA6T-1682M?

A: We observed a full 2X improvement over 1-core operation

- Two copies of our most intensive benchmarking application ran on two cores at 1.7 GHz with no interference or extra overhead
- Application included significant VMX (AltiVec™) use and significant main-memory utilization



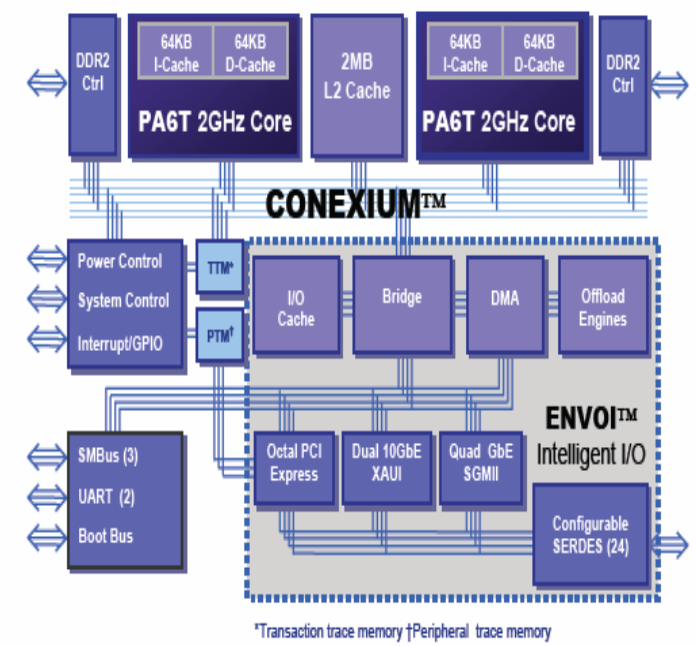
Used with permission from P.A. Semi for HPEC 2007

Multi-core operation
Dual-core power management

Q: How is power consumption affected by dual-core operation?

A: Power consumption was "as advertised"

- **Power-management also allowed the cores to drop to lower clock speeds (and much lower power consumption) when processing loads were removed**
  - **Both cores operate at the same speed**
  - **Core power consumption is largely determined by clock speed**
    - **With clock forced to 1.7 GHz, we measured 11.5 watts with no application running**
    - **Drops to 3.6 watts at 400 Hz if permitted**
  - **Our most intensive benchmarking application running full speed on both cores at 1.7 GHz never exceeded 11.9 watts**

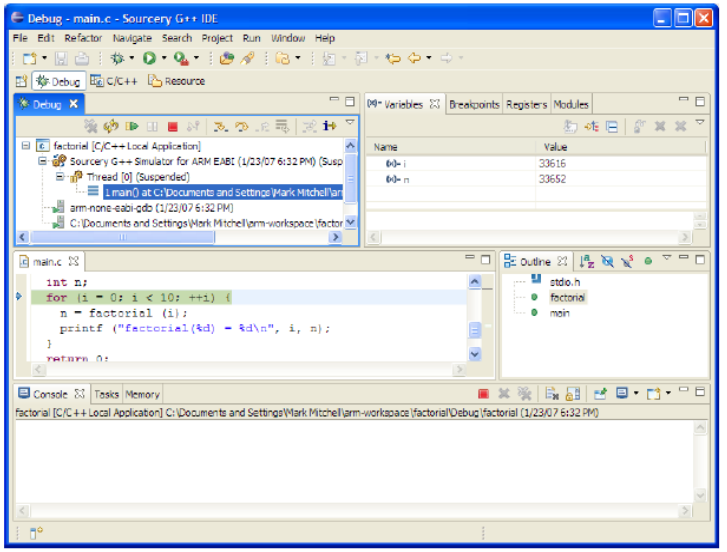


Used with permission from P.A. Semi for HPEC 2007

Software productivity

Use of Sourcery G++ IDE

- Eclipse-based IDE, GNU toolchain
- Productive source-code development on desktop/laptop computers
- EABI simulator
  - application code to run and debugged on without leaving the IDE
  - VSIPL++ and Mercury CSAL (C-language version of SAL) easily compiled for use with the simulator
- Ease-of-use significantly contributed to software development productivity



- G++ IDE on Host platform (networked to target platform)
  - IDE supports interactive debugging on target processor (Electra w/ PA6T-1682M) without leaving IDE on host
  - Transition from development platform target platform nearly “seamless”

Software productivity
Use of VSIPL++ and SAL

- Project team had no prior experience with VSIPL++ or C++, some experience with SAL, significant experience with C-language
- Two months to learn a new language and library and build code is a challenge
  - Support from CodeSourcery was invaluable
  - One gripe: VSIPL++ needs better documentation!
- Objective results
  - Ported 1250 lines of unfamiliar C/SAL source code to 850 lines of working C++/VSIPL++ code in 4 programmer-weeks
    - Use of VSIPL++ led to compact code
  - In an additional 1.5 programmer-weeks, added 400 additional lines to permit program to be run in pure VSIPL++ API configuration or VSIPL++/SAL API configuration
  - More than half of this additional code was routine “glue” to translate between VSIPL++ objects and C arrays which SAL could access

Software productivity
Use of VSIPL++ and SAL: Subjective comments

- VSIPL++ and object-oriented design contributed to productivity
  - Easy to read code – relatively few chances for error
  - Hence greater productivity
- Productivity was enhanced by VSIPL++
  - Syntax takes care of much of the vector-matrix "bookkeeping"
  - Error-prone code (*e.g.,* index management for subviews of matrices) is simplified
- VSIPL++ and SAL jointly supported a productive approach to building high-performance code
  - First, a clean design with correct operation
  - Then optimization in selected areas (*e.g.,* by insertion of SAL code) based on profiling information (in our case, from use of VSIPL++ Profiling API)
- Inserting SAL at lower levels was easy once design was in place
  - Clean design was preserved
  - Bookkeeping glue (managing index bounds, etc.) was easy to write

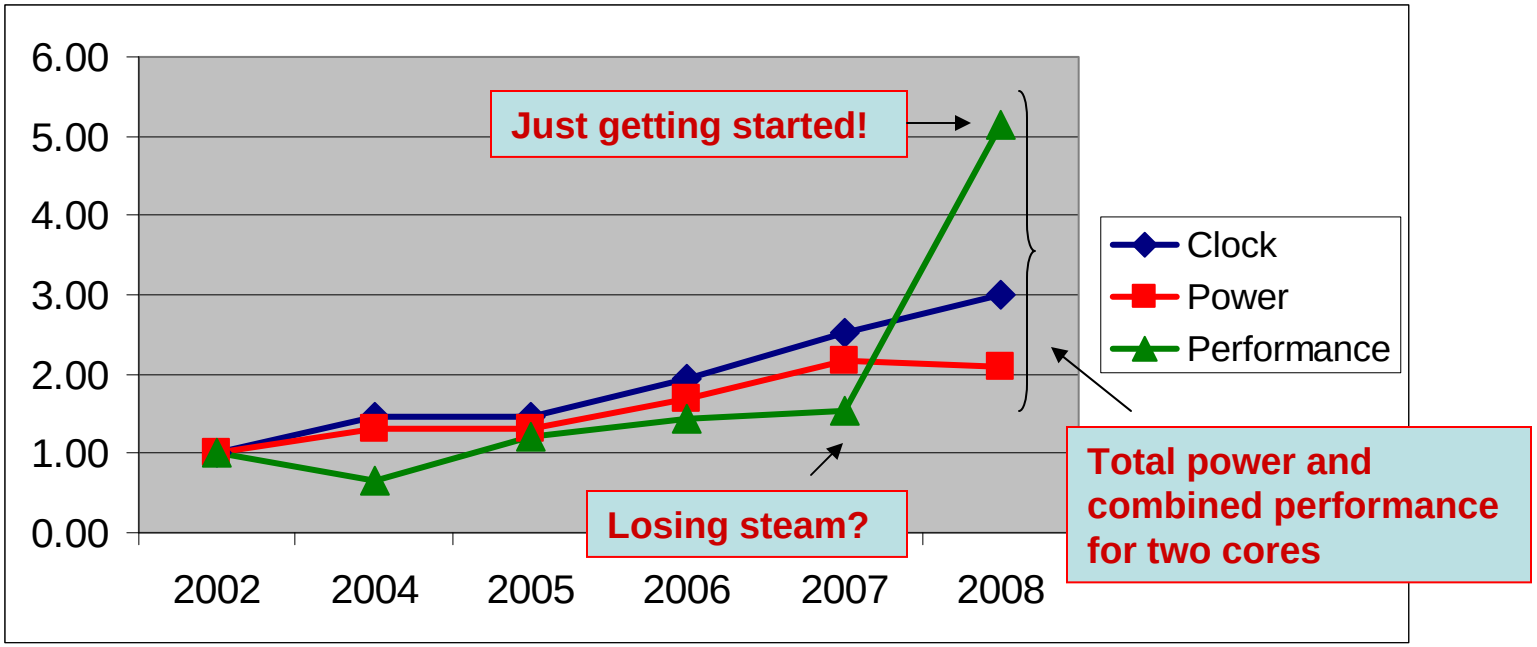
Software portability
Benchmarking the PA6T-1682M

- Software portability was excellent!
- Both VSIPL++ and SAL application code ported seamlessly from desktop (using CSAL) to PA6T-1682M Target
  - Binary-compatible between IBM 970 FX and PA6T-1682M
    - Both had Linux operating systems
    - Code compiled for generic Power Architecture™ with VMX
    - Source-compatible (re-compilation req'd) with other targets
      - 7447A in Mercury Powerstream™ system with MCOE operating environment
      - Sourcery G++ EABI power-simulator environment on desktop without AltiVec™ instruction support
- There was never a reason to run debugger on PA6T-1682M target!
  - Once software ran correctly on desktop, it also always ran correctly on PA6T-1682M

Software performance
Including use of VSIPL++ and SAL

- Software performance on our benchmarks relied on no processor-specific optimizations
  - There was no explicit control of caches
  - Source-code optimization consisted only of changes to make effective use of VSIPL++ and SAL libraries
- Sourcery VSIPL++ uses many mechanisms to achieve portable performance
  - An important one in the present context is linking to SAL
- We achieved best performance on benchmarks by explicit use of SAL in our applications
  - Another mechanism used was to improve Sourcery VSIPL++ dispatching to make better use of SAL
  - Project schedule did not permit taking this latter approach to its logical conclusion to minimize the performance gap between explicit (application level) and implicit (library linking) use of SAL
  - Purpose of such an approach would be to maximize portability with minimal sacrifice of performance

Wrap-up: Conservative prediction
Based on already-demonstrated performance



- **Recall the disappointing graph on Slide 3**
- **Here is a VERY CONSERVATIVE projection based on already-demonstrated performance, BUT (see next slide)…**

Wrap-up: The best is yet to come
We've barely scratched the surface!

- With:
  - PA6T-1682M production silicon coming soon
    - Testing was on A.2 pre-production silicon
  - Mercury SAL libraries ready to be optimized for PA6T
    - Testing was done using un-optimized libraries
  - Sourcery VSIPL++ continuing to be improved
- Significant additional performance improvements by this time next year are:
  - Not just likely, but
  - Near-certain
- We expect to see at least a factor of 1.3x to 1.5x additional improvement

Anyone want to place bets?
See you at HPEC 2008!

Exploring Multi-core Processors using Realistic Signal- and Image-processing Application Benchmarks
-Acknowledgements-

- This presentation is based on ongoing Independent Research and Development (IRAD) being conducted by Lockheed Martin Corporation
- This presentation was prepared by Lockheed Martin MS2 Tactical Systems (Eagan) with contributions from:
  - Other Lockheed Martin organizations
- We gratefully acknowledge significant contributions, including presentation co-authorship, from:
  - CodeSourcery, Inc.
- We also gratefully acknowledge significant technical assistance (but no responsibility for presentation content) from:
  - Mercury Computer Systems, Inc.
  - P.A. Semi, Inc.