Processor families in TOP500 supercomputers

Intel Itanium

Matt Layman
Adam Sanders
Aaron Still
Overview

- History
  - 32 bit Processors (Pentium Pro, Pentium Xeon)
  - 64 bit Processors (Xeon, Itanium, Itanium 2)
- ISA
  - EPIC
  - Predicated Execution (Branch Prediction)
  - Software Pipelining

Overview

- ISA cont.
  - Register Stacking
  - IA-32 Emulation
  - Speculation
- Architecture
- Benchmarks
History

- 32 bit processors
  - Pentium Pro
    - Based on P6 core
    - 256 kB – 1 MB L2 cache
    - Optimized for 32 bit code
    - x86 ISA
    - L2 cache was “on-package,” bonded to die before testing (low yields, high costs)

- 32 bit processors
  - Pentium II Xeon
    - Server replacement for Pentium Pro
    - Roughly comparable specs to Pro
  - Pentium III Xeon
    - Based on Pentium III core
    - L2 cache moved on die
    - Supports SSE
History

- 32 bit processors
  - Xeon
    - Based on Pentium 4 Netburst architecture
    - Hyperthreading support
    - SSE2 support
    - L3 cache added (1 – 2 MB)

History

- 64 bit processors
  - Xeon
    - Based on Pentium 4 Netburst architecture
    - SSE3 support
    - EM64T ISA (Intel’s name for AMD64)
    - Contains execute disable (XD) bit
History

- 64 bit processors
  - Itanium (1)
  - Itanium 2

- Itanium (1)
  - Code Name: Merced
    - Shipped in June of 2001
    - 180 nm process
    - 733 / 800 MHz
    - Cache 2 MB or 4 MB off-die
    - The only version of Itanium 1
History

- The Itanic - Original Itanium was expensive and slow executing 32 bit code
History

- French translation:
  “It’s back and it’s not happy”
  (loose translation)

- Itanium 2
- Common Features: 16 kB L1 I-cache, 16 kB L1 D-cache, 256 kB L2 cache
- Revisions:
  - McKinley, Madison, Hondo, Deerfield, Fanwood
- Upcoming Revisions:
  - Montecito, Montvale, Tukwila, Poulson
History

- Itanium 2
  - Code Name: McKinley
    - Shipped in July of 2002
    - 180 nm process
    - .9 / 1 GHz
    - L3 Cache 1.5 / 3 MB respectively
History

- Itanium 2
  - Code Name: Madison
    - Shipped in June of 2003
    - 180 nm process
    - 1.3 / 1.4 / 1.5 GHz
    - L3 Cache 3 / 4 / 6 MB respectively

- Itanium 2
  - Code Name: Hondo
    - Shipped early 2004 (only from HP)
    - 2 Madison cores
    - 180 nm process
    - 1.1 GHz
    - 4 MB L3 cache each
    - 32 MB L4 cache shared
History

**Itanium 2**
- Code Name: Deerfield
  - Released September 2003
  - 1st low voltage Itanium suited for 1U servers
  - 180 nm process
  - 1 GHz
  - L3 Cache 1.5 MB

**History**

**Itanium 2**
- Code Name: Fanwood
  - Release November 2004
  - 180 nm process
  - 1.3 /1.6 GHz
  - L3 Cache 3 MB in both chips
  - 1.3 GHz is a low voltage version of the Fanwood
History

- Itanium 2
  - Code Name: Montecito
    - Expected Release in Summer 2006 (recently delayed)
    - Multi-core design
    - Advanced power and thermal management improvements
    - Coarse multi-threading (not simultaneous)

History

- Itanium 2
  - Code Name: Montecito
    - 90 nm process
    - 1 MB L2 I-cache, 256 kB L2 D-cache
    - 12 MB L3 cache per core (24 MB total)
    - 1.72 billion transistors per die (1.5 billion from L3 cache)
Introducing Montecito

- 2 Way Multi-threading
- Power Management/Frequency Boost (Fiston)
- Dual-core
- 1MB L2 caches
- Arbiter
- 2x12MB L3
- Soft Error Detection/Correction
- with Pelston

http://www.pcmag.com/article2/0,4149,222505,00.asp

Now it’s time for some Intel Propaganda...

http://mfile.akamai.com/10430/wmv/
cim.download.akamai.com/10430/biz/
itanium2_everyday_TI.asx
“Intel has not verified any of these results”

ISA Overview

- Most Modern Processors:
  - Instruction Level Parallelism (ILP)
  - Processor, at runtime, decides which instructions have no dependencies
  - Hardware branch prediction
Itanium’s ISA

- IA-64 – Intel’s (first) 64-bit ISA
- Not an extension to x86 ( Completely new ISA)
- Allows for speedups without engineering “tricks”
- Largely RISC
- Surrounded by patents
IA-64

- IA-64 largely depends on software for parallelism
- VLIW – Very Long Instruction Word
- EPIC – Explicitly Parallel Instruction Computer

IA-64

- VLIW – Overview
  - RISC technique
  - Bundles of instructions to be run in parallel
  - Similar to superscaling
  - Uses compiler instead of branch prediction hardware
IA-64

- EPIC – Overview
  - Builds on VLIW
  - Redefines instruction format
  - Instruction coding tells CPU how to process data
  - Very compiler dependent
  - Predicated execution

“The compiler is essentially creating a record of execution; the hardware is merely a playback device, the equivalent of a DVD player for example.”

D'Arcy Lemay

http://www.devhardware.com/index2.php?option=content&task=view&id=1443&pop=1&page=0&hide_j s=1
Instruction Format: Bundles & Templates

- Bundle (123 bits)
  - Set of three instructions
- Template (5 bits)
  - Identifies types of instructions in bundle
    - One of Integer, Memory, Branch, Floating, eXtended
  - Identifies independent operations ("stops") -> MM_F
  - Defines execution units to be invoked executing the bundle
  - Compiler can schedule functional units to avoid contention

Explicitly Parallel Instruction Computing
EPIC

- Fetch one or more bundles for execution (Implementation, Itanium® takes two.)
- Try to execute all instructions in parallel, depending on available units.
- Retired instruction bundles
Predicated Execution:
- Decrease need for branch prediction
- Increase number of speculative executions
- Branch conditions put into predicate registers
- Predicate registers kill results of executions from not-taken branch
Predicated Instructions

Requires
- Hardware
- ISA modification

Predicated instructions eliminate branches, converting a control dependence into a data dependence.

IA-64 has predicated instructions, but many existing ISA contain at least one (the conditional move).

Conditional Move

if (R1 == 0) R2 = R3;

Branch:
  BNEZ R1,L
  ADDU R2, R3, R0

L:

Conditional Move:
  CMOVZ R2, R3, R1

In a pipeline, the control dependence at the beginning of the pipeline is transformed into a data dependence at the end of the pipeline.
Predicated Execution

Convert control flow dependency to data dependency

**Pro:**
- Eliminate hard-to-predict branches

**Cons:**
- (1) Fetch blocks B and C all the time
- (2) Don’t commit until p1 is resolved

Predication

- Use predicates to eliminate branches, move instructions across branches
- Conditional execution of an instruction is based on predicate register
- Predicates are set by compare instructions
- Most instructions can be predicated – each instruction code contains predicate field
- If predicate is true, the instruction updates the computation state; otherwise, it behaves like a NOP
Full Predication

Every instruction has a predicate:
if the predicate is false, it becomes a NOP.

It is particularly useful for global scheduling
since non-loop branches can be eliminated:
the harder ones to schedule.

Exceptions & Predication

A predicated instruction must not be allowed
to generate an exception,
if the predicate is false.
Implementation

Although predicated instructions can be annulled early in the pipeline, annulling during commit delays annullment until later so data hazards have an opportunity to be resolved.

The disadvantage is that resources such as functional units and registers (rename or other) are used.

Predication is good for…

- Short alternative control flow
- Eliminating some unpredictable branches
- Reducing the overhead of global scheduling

However, the precise rules for compilation have turned out to be elusive.
Predication

- One-bit predicate registers are used as a switch (status: on or off) and indicate whether a particular branch is “turned on” or “turned off”.
- Branches which are ‘turned on’ continue to execute, and branches which are ‘turned off’ cease to execute.
- It’s an efficient way to reduce branches and associated miss-predicts without the cost of adding many instructions.
- Implementation?

Itanium

- Every instruction has a predicate.
- There exists logic for the predicates so predicates can be logical combinations of other predicates.
  - Therefore, complicated logic for a branch can be reduced to one predicate register.
  - Also, branches which depend on other branches can be similarly reduced.
- Itanium has 64 one-bit predicate registers.
Limitations

- Anulled instructions waste resources: registers, functional units, cache & memory bandwidth.
- If predicate condition cannot be separated from the instruction, a branch might have had better performance, if it could have been accurately predicted.

Limitations (con’t)

- Predication across multiple branches can complicate control and is undesirable unless hardware supports it (as in IA-64).
- Predicated instructions may have a speed penalty—not the case when all instructions are predicated.
Hoisting Loads

Hoist Load Example

if (A==0) A=B; else A= A+4;

LD R1,0(R3); load A
BNEZ R1,L1; test A
LD R1,0(R2); then clause
J L2; skip else
L1: DADDI R1,R1,#4; else clause
L2: SD R1,0(R3); store A
**Hoist Load**

if \((A == 0)\) \(A = B\); else \(A = A + 4\);

- \(\text{LD} \ R1,0(R3)\); load \(A\)
- \(\text{LD} \ R14,0(R2)\); speculative load \(B\)
- \(\text{BEQZ} \ R1,L3\); other branch of if
- \(\text{DADDI} \ R14,R1,#4\); else clause
- \(L3: \text{SD} \ R14,0(R3)\); store \(A\)

What if speculative load raises an exception?

**Guard**

if \((A == 0)\) \(A = B\); else \(A = A + 4\);

- \(\text{LD} \ R1,0(R3)\); load \(A\)
- \(\text{sLD} \ R14,0(R2)\); speculative load
- \(\text{BNEZ} \ R1,L1\); test \(A\)
- \(\text{SPECCK} \ 0(R2)\); speculative check
- \(J \ L2\); skip else
- \(L1: \text{DADDI} \ R14,R1,#4\); else clause
- \(L2: \text{SD} \ R14,0(R3)\); store \(A\)

\(\text{sLD}\) does not raise certain exceptions; leaves them for \(\text{SPECCK}\) (IA-64).
Other exception techniques

- Poison bit:
  - applied to destination register.
  - set upon exception
  - raise exception upon access to poisoned register.

Hoist Load above Store

If memory addresses are known, a load can be hoisted above a store.

If not, …

add a special instruction to check addresses before the loaded value is used.

(It is similar to SPECCK shown earlier: IA-64)
Speculation: soft vs. hard

- must be able to disambiguate memory (to hoist loads past stores), but at compile time information is insufficient
- hardware works best when control flow is unpredictable and when hardware branch prediction is superior
- exception handling is easier in hardware
- trace techniques require compensation code
- compilers see further for better scheduling

Predication

Traditional Architecture  Itanium® Architecture

Code for both paths loaded and routed to different execution pipelines.
Only one ‘branch’ will have a valid predicate and be executed.
IA-64

- Software Pipelining:
  - Take advantage of programming trends and large number of available registers
  - Allow multiple iterations of a loop to be in flight at once
IA-64

- Register Stacking:
  - First 32 registers are “global”
  - Create “frame” in next higher registers for procedure-specific registers
  - When calling procedures, rename registers and add new local variables to top of “frame”
  - When returning, write outputs to memory, but restore state by renaming registers – much faster

Register Stack
- Call changes the frame to contain only the caller output
- Alloc sets the frame region to the desired size
  - Three architecture parameters: local, output, and rotating
- Return restores the stack frame of the caller

Avoid Register Spill/Fill Upon Procedure Call/Return
IA-64

EPIC – Pros:
- Compiler has more time to spend with code
- Time spent by compiler is a one-time cost
- Reduces circuit complexity

IA-64

EPIC – Cons:
- Runtime behavior isn’t always obvious in source code
- Runtime behavior may depend on input data
- Depends greatly on compiler performance
IA-64

- IA-32 Support:
  - Done with hardware emulation
  - Uses special jump escape instructions to access
  - Slow (painfully so)

IA-64

- 32 Bit Hardware Emulation – Very Poor Performance

- Software Emulation of x86 32-bit from either Microsoft or Linux can perform 50% better than Intel’s Hardware Emulation

- Less than 1% of the chip devoted to Hardware Emulation
IA-64

- On 32 Bit Hardware Emulation, “Tweakers.net finds that the 32bit hardware portion of a 667Mhz Itanic wheezes along at the speed of a 75Mhz Pentium.”

Andrew Orloski

http://www.theregister.co.uk/2001/01/23/benchmarks_itanic_32bit_emulation/

IA-64

- IA-32 Slowness:
  - No out-of-order execution abilities
  - Functional units don’t generate flags
  - Multiple outstanding unaligned memory loads not supported
IA-64

- IA-32 Support:
  - Hardware emulation augmented for Itanium 2
  - Software emulation (IA-32 Execution Layer) added
  - Runs IA-32 code at same speed as equivalently clocked Xeon

IA-64

- Data Speculation:
  - Loads/stores issued in advance of their occurrence (when instruction bundles have a free memory slot)
  - Keeps memory bus occupied
  - For failed speculation, load/store issued when it normally would have (no real loss)
IA-64

- Code Speculation:
  - Instructions issued speculatively to otherwise unused functional units
  - Results not written back (kept in a temporary area) until execution of those instructions is valid
  - Exceptions are deferred (to ascertain if the instruction should have ever been executed)

Overview from Tuesday

- History
  - Itanium is 64 bit (duh!) – if we failed to communicate that to you, we failed miserably

- ISA
  - VLIW / EPIC
  - Predicated Execution
Architecture

- Physical Layout
- Conceptual Design Elements

Byte Ordering

- All IA-32 are little Endian
- IA-64 is little Endian by default
Alignment

- Data item size = 1, 2, 4, 8, 10, 16 bytes
- Intel suggestions are “recommended for optimum speed” (read: do it this way, or don’t blame Intel for poor performance)

Large Constants

- Instructions fixed at 41 bits
- Constants limited to 22 bits
- But actually constants have 63 bits. How?
Memory Addressing

- Original Itanium addressed – $2^{36}$ bits (64 GB)
- McKinley and later (Itanium 2) – $2^{44}$ bits (18 TB)

How big is an EB?

- In zeroes:
  
  00000000000000000000000000000000000000000000000000000000000000000000000
  00000000000000000000000000000000000000000000000000000000000000000000000
  00000000000000000000000000000000000000000000000000000000000000000000000
  ...

- This is only $2^{10}$ digits. 1 EB would be $2^{54}$ times bigger than this.
Registers

- 128 82-bit Floating Point Registers
  - 1 bit sign, 17 bit exponent, 64 bit mantissa
- 128 64-bit General Purpose Registers
- 64 1-bit Predicate Registers
- 8 64-bit Branch Registers
  - Used to hold indirect branching information
- 8 64-bit Kernel Registers

 Registers

- 1 64-bit Current Frame Marker (CFM)
  - Used for stack frame operations
- 1 64-bit Instruction Pointer (IP)
  - Offsets to one byte aligned instruction OR
    holds pointer to current 16 byte aligned bundle
 Registers

- 256 1 bit NaT and NaTVal registers (Not a Thing)
  - Indicates deferred exceptions in speculative execution
- Several other 64 bit registers

 Register File

- Floating Point Registers
  - 8 read ports
  - 4 write ports
- General Purpose Registers
  - 8 read ports
  - 6 write ports
Register File

- Predicate Registers
  - 15 read ports
  - 11 write ports

Register Stack Engine (RSE)

- Improve performance by removing latency associated with saving/restoring state for function calls

- Hardware implementation of register stack ISA functionality
Itanium Pipeline

- 10 Stage
  - Instruction Pointer Generation
  - Fetch
  - Rotate
  - Expand
  - Rename
  - Word-Line Decode
  - Register Read
  - Execute
  - Exception Detect
  - Write Back
Itanium 2 Pipeline

- 8 stage
  - Instruction Pointer Generation
  - Rotate
  - Expand
  - Rename
  - Register Read
  - Execute
  - Detect
  - Write Back
Processor Abstraction Layer (PAL)

- Internal processor firmware
- External system firmware
Parallel EPIC Execution Core

- 4 Integer ALUs
- 4 Multimedia ALUs
- 2 Extended Precision FP Units
- 2 Additional Single Precision FP Units
- 2 Load / Store Units
- 3 Branch units

- 6 instructions per clock cycle
Instruction Prefetch and Fetch

- Speculative fetch from instruction cache
- Instruction go to decoupling buffer
  - Hides instruction cache and prediction latencies
- Software-initiated prefetch

I-Cache

- 16KB
- 4-way set-associative
- Fully pipelined
- 32B deliverable (6 instructions in 2 bundles)
- I-TLB
  - Fully associative
  - On-chip hardware page walker
Branch Prediction

- 4 way hierarchy
  - Resteer1: Special single-cycle branch predictor
  - Resteer2: Adaptive two-level multi-way predictor
  - Resteer3-4: Branch address calculate and correct

- Itanium 2: Simplified
  - 0-bubble branch prediction algorithm with a backup branch prediction table.
Instruction Disperse

- 9 issue ports
  - 2 memory instruction
  - 2 integer
  - 2 floating-point
  - 3 branch instructions

Itanium 2 – 11 issue ports

Q: How many Intel architects does it take to change a lightbulb?

A: None, they have a predicated compiler that eliminates lightbulb dependencies. If the dependencies are not entirely eliminated, they have four levels of prediction to determine if you need to replace the lightbulb.
Decoupling Buffer

- Hides latency from cache and misprediction
  - Disperses instructions to pipeline
  - Granular dispersal

Itanium Execution Core

- 4 ALU
- 4 MMX
- 2 + 2 FMAC
- 2 Load / Store
- 3 branch
**Itanium 2 Execution Core**

- 6 multimedia units
- 6 integer units
- 2 FPU
- 3 branch units
- 4 load / store units

**Data Dependencies**

- Register Scoreboard
  - Hazard detection
  - Stall on dependency
  - Deferred stalling
Floating Point Unit

- Independent FPU register file (128 entry)
  - 4 write, 8 read
- 6.4 Gflops throughput
- Supports single, double, extended, and mixed mode precision
- Can execute two 32bit single precision numbers in parallel
- Pipelined
Control

- Exception handler
  - Exception prioritizing
- Pipeline control
  - Based on the scoreboard, supports data speculation as well as predication

Memory Subsystem

[Diagram showing the Memory Subsystem with components like L1, L2, L3, Bus Logic, 64b Frontside Bus, 128b Backside Bus, and Itanium Processor Cartridge]
Advanced Load Address Table

- Data speculation
- 32 entries
- 2-way set-associative

IA-32 Execution Hardware

IA-32 Instruction Fetch and Decode → IA-32 Dynamic Scheduler → IA-32 Retirement & Exceptions → Shared Instruction Cache and TLB → Shared Itanium™ Processor Execution Core
Improving ILP

- Improving execution
  - New instructions
  - Additional functional units
  - More efficient speculation recovery
- Improving the memory hierarchy
  - Split the L2 cache
    - Dedicated 1 MByte L2 instruction cache
    - Reduced pressure on L2 data cache
  - Larger L3
    - Grown L3 to 12 MByte
    - Maintain L3 latency of Itanium® 2 processor 64M and 9M
  - Queues and Control
    - Additional L3 and L2 victim buffers
    - More efficient control of queues and coherence

Error Protection

- Protection is adapted to the structure
- Performance structures are covered for lockstep

Intel®

System Interface

Branch Pattern Table
Branch History Table
Integer Registers
Floating Point Registers

L1I 16KB
Tag
Data
L1D 16KB
Tag
Data
L2I 1MB
Tag
Data
LRU
L2D 256KB
Tag
Slate
Data
L3 12MB
Unified

ECC
Parity
Multiple Hit
1 From Modified
Introducing Montecito

Dual-core

1MB L2

2 Way Multi-threading

2x12MB L3 caches with Pelton

Soft Error Detection/Correction

Arbiter

Power Management/ Frequency Boost (Foxton)

http://www.pcmag.com/article2/0,4149,222505,00.asp

Montecito Arbiter

System Interface Controller

System Interface

Snoop Data

Write Data

Write Requests

Caselout Requests

Read Requests

Snoop Controller

Core 0

Core 1

Low latency, high throughput single interface for two cores
Itanium® 2 Performance

- Integer
- Technical
- Enterprise

Montecito*
- 90nm

Itanium® 2 9M *
- 130nm

Itanium 2 6M
- 130nm

Itanium 2
- 180nm

Performance with multiple cores, multiple threads, larger caches, and higher frequency

---

**Benchmarks**

<table>
<thead>
<tr>
<th></th>
<th>Revision 2</th>
<th>Revision 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fabrication Process</td>
<td>0.18 micron</td>
<td>0.18 micron</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>900, 1000 MHz</td>
<td>730, 800 MHz</td>
</tr>
<tr>
<td>System Bus Bandwidth</td>
<td>6.4 GB/Sec</td>
<td>5.1 GB/Sec</td>
</tr>
<tr>
<td>(667 MHz, 2400ns Wide)</td>
<td>(533 MHz, 2400ns Wide)</td>
<td></td>
</tr>
<tr>
<td>Level 1 Cache [Instruction/Class]</td>
<td>16 / 16</td>
<td>16 / 16</td>
</tr>
<tr>
<td>Level 2 Cache</td>
<td>226 KB</td>
<td>96 KB</td>
</tr>
<tr>
<td>Level 3 Cache</td>
<td>1.5 MB or 3 MB (On Chip)</td>
<td>2 MB or 4 MB (Off Chip, On Board)</td>
</tr>
<tr>
<td>Pipeline Stages</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>Number of Issue Pairs</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>Total Number of Registers</td>
<td>323</td>
<td>323</td>
</tr>
<tr>
<td>Total Number of Execution Units</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Fetch Instructions Per Cycle</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Processor Price (in Quantities of 1000)</td>
<td>$1,335</td>
<td>$1,177</td>
</tr>
<tr>
<td>1000 MHz</td>
<td>$4,235</td>
<td>$3,161</td>
</tr>
<tr>
<td>800 MHz</td>
<td>$4,227</td>
<td></td>
</tr>
</tbody>
</table>


2002
### Benchmarks

IDEAS Top Performers - SPECfp2000 - Single CPU Subset

<table>
<thead>
<tr>
<th>Rank</th>
<th>Company</th>
<th>System</th>
<th>Processor</th>
<th>Peak Result</th>
<th>Baseline</th>
<th>Test Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HITACHI</td>
<td>HITACHI BladeSymphony (1.66GHz/5MB, Itanium 2)</td>
<td>Intel Itanium 2</td>
<td>2901</td>
<td>Jun-05</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Hewlett-Packard</td>
<td>HP Integrity rx4640-6 (1.5GHz/9MB, Itanium 2)</td>
<td>Intel Itanium 2</td>
<td>2712</td>
<td>Oct-04</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Hewlett-Packard</td>
<td>HP Integrity rx1620-2 (3.0GHz/3MB, Itanium 2)</td>
<td>Intel Itanium 2 (1.6GHz/3MB, 533 MHz FSB)</td>
<td>2692</td>
<td>Oct-04</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Hewlett-Packard</td>
<td>HP Integrity rx1620-2 (3.0GHz/3MB, Itanium 2)</td>
<td>Intel Itanium 2 (1.6GHz/3MB, 400MHz FSB)</td>
<td>2675</td>
<td>Dec-04</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SGI</td>
<td>SGI Altix 3700 8x2 (1400MHz 6M, Itanium 2)</td>
<td>Intel Itanium 2</td>
<td>2647</td>
<td>Oct-04</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SGI</td>
<td>SGI Altix 3700 8x2 (1400MHz 6M, Itanium 2)</td>
<td>Intel Itanium 2</td>
<td>2600</td>
<td>Oct-04</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>IBM Corporation</td>
<td>IBM eServer p5 595 (1GHz, 1 CPU)</td>
<td>POWER5</td>
<td>2756</td>
<td>Jan-05</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>IBM Corporation</td>
<td>IBM eServer p5 599 (1GHz, 1 CPU)</td>
<td>POWER5</td>
<td>2702</td>
<td>Jun-04</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Sun Microsystems</td>
<td>Sun Fire X4400</td>
<td>AMD Opteron (TM) 254</td>
<td>2518</td>
<td>Aug-05</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Sun Microsystems</td>
<td>Sun Fire X4400</td>
<td>AMD Opteron (TM) 254</td>
<td>2518</td>
<td>Aug-05</td>
<td></td>
</tr>
</tbody>
</table>


---

### Benchmarks

**SGI® Altix™ 3000**

**Intel Itanium® 2-based (Madison) Benchmark Results**

**June 2003**

**128P Tests**

<table>
<thead>
<tr>
<th>System</th>
<th>Linpack N/s</th>
<th>IBM eServer p690</th>
<th>HP Superdome™</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGI® Altix™ 3000</td>
<td>653.3 Gflops</td>
<td>378.2 Gflops</td>
<td>248.9 Gflops</td>
</tr>
</tbody>
</table>

2x64P Intel Itanium 2 processor 1.30 GHz with 3M L3 cache, NUMA link interconnect

**64P Tests**

<table>
<thead>
<tr>
<th>System</th>
<th>STREAM Triad</th>
<th>Sun Fire™ 15K</th>
<th>HP Superdome™</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGI® Altix™ 3000</td>
<td>127 GB/sec</td>
<td>51* GB/sec</td>
<td>27 GB/sec</td>
</tr>
</tbody>
</table>

**Using 72 CPUs 1050 MHz UltraSPARC III CPU, no 64P results available**

www.cs.wisc.edu/stream/

Solid Mission Critical Roadmap

Today 2013 Future

Modular Development Model

Sustainable roadmap for the long-term future

Poulson Itanium 9500
Poulson 9500

- 89 million transistors per core (down from 109 million in Tukwilas)
- 8 cores
- 12-wide instruction issue
- 32 nm process
- 2.53 GHz clock (46% faster than Tukwilas)
- 80% less idle power

Fail safe

- Automatically retry instructions instead of failing and crashing app or system
- Blocks out bad areas of memory hierarchy
- QuickPath Interconnect 6.4 GB/sec connects processors
- Turbo Boost “dynamically allocates power to the area where power is needed most”

---

**Performance**

- TPC-C
  - [http://www.tpc.org/tpcc/results/tpcc_perf_results.asp](http://www.tpc.org/tpcc/results/tpcc_perf_results.asp)
- TPC-H
  - [http://www.tpc.org/tpch/results/tpch_perf_results.asp](http://www.tpc.org/tpch/results/tpch_perf_results.asp)