HP WORLD 2004
Solutions and Technology Conference & Expo

Optimizing Application Performance on Itanium2/Linux Servers:

Compilers, Libraries, Tools and Case Studies

Logan Sankaran, Ph.D.

Sr. Solution Architect, Americas Solution Center
Hewlett-Packard

Overview

- Intel Compilers for Itanium and Linux
- Base Optimizations
- Advanced Optimization Techniques
- Parallel Computing Support
- Optimization Report
- Libraries and Tools
- Results and Comparison
- Conclusions

Intel Compilers …

- Unified Compilers
  - ecc for C and C++ Languages
  - efc for Fortran77, Fortran90 and Fortran95
- Compatibility with Standards
  - ecc is compatible with Linux gcc
  - efc supports various Fortran, OpenMP and MPI standards
- Ease of USE and Efficiency
  - Automatic Optimization Features
  - Automatic Parallelization Support

Intel Compilers …

- Advanced Optimization Features
  - Inter-Procedural Optimization (IPO)
  - Profile-Guided Optimization (PGO)
- Parallel Computing Support
  - Multi-threading using OpenMP directives
  - Support through POSIX thread libraries
  - Multi-processing using Message Passing Interface (MPI)
- Outstanding Optimization Report
  - Options to generate report at various degree of detail
  - Optimization phase based compiler reports

Intel Compilers

- Standard Libraries Support
  - Large collection of shared and archived libraries
  - Libraries for portability, functionality and compatibility
  - Support for Math Kernel Library (MKL) and Intel Integrated Performance Primitives (IPP)
- Tools for Porting and Optimization
  - GNU source level debugger (gdb)
  - Intel's Linux debugger (ldb)
  - Vtune Performance Analyzer for Sampling and Profiling
- Intel Premier Support
  - Web-based problem reporting and tracking
  - Software, patches, bug fixes and beta products download
  - Training and Education through Intel Software College

Basic Compiler Flags …

–help : print detailed help message

–V : display compiler version information (-logo)

–i<M>: set default size of INTEGER to <M> (=2, 4, 8)

–r<N>: set default size of REAL to <N> (=8, 16)

–integer_size <size>: set default size of integer and logical variables to <size> (=16, 32, 64)

–real_size <size> : set default size of real and complex declarations, constants, functions and intrinsics to <size> (=32, 64, 128)

–DD : compile debug statements, indicated by D in column 1

–FR : specifies source files are in FREE format

–FI : specifies source files are in FIXED format

– <NN>: specifies <NN> (=72,80,132) column lines for FIXED form sources

Basic Compiler Flags

–save : save all variables (static allocation)

–auto : make all local variables AUTOMATIC (opposite of –save)

–u, -implicitnone : set IMPLICIT NONE by default

–list : print source listing on stdout

–cm : suppress all comment messages

–q : suppress printing errors to stderr

–w : disable all warning messages

–Wn : disable warnings n=0; show warnings n=1 (default)

–WB : issue warnings not errors for array bound checking

–w90, -w95 : suppress warnings for non-standard Fortran

–e90, -e95 : issue errors for non-standard Fortran

–[no]warn <keywords> : [do not] issue warning messages for items indicated by <keyword> (= alignments, stderrors)

HP WORLD 2004
Solutions and Technology Conference & Expo

Base Optimizations …

- Choice of Opt. Levels controlling Speed and Accuracy

–O0 : Good starting point for a new application
Disables most of the optimizations

–O1 : Omits optimizations that increase code size
Creates smallest code in most cases

–O2 : Default setting and same as –O (compatibility)
Enables optimizations including function inlining
Creates fastest code in most cases
May increase code size significantly over –O1

–O3 : Builds over –O2 plus more aggressive optimizations
such as: loop unroll, loop blocking, loop reordering, data prefetching, pipelining etc.

Base Optimizations…

–falias	: assume aliasing in a program (default)
–ffnalias	: assume aliasing within functions (default)
–fno-alias	: assume no aliasing in a program
–fno-fnalias	: assume no aliasing within functions but assume aliasing across calls
–mp	: maintain floating point precision
–mp1	: improve floating point precision (speed impact is less than that of –mp)
–ftz[-]	: enable/disable flushing denormalized results to zero (may impact accuracy)

Base Optimizations…

–tpp1 : optimize for Itanium processor

–tpp2 : optimize for Itanium2 processor (default)

–mcpu=<cpu> : optimize for specific cpu:

itanium – optimize for Itanium

itanium2 – optimize for Itanium2

–Ob<n> : control inline expansion (ecc only)

n = 0 disable inlining

n = 1 inline functions declared with __inline and C++ inlining

n = 2 inline any function at the discretion of the compiler

HP WORLD 2004 Solutions and Technology Conference & Expo

Base Optimizations

–fpe{0|1|2} : specifies behavior on FP exceptions

–[no]fltcpnsistency : enable [disable] improved floating point consistency

–nolib_inline : disable inline expansion of intrinsic functions

–pad : enable changing variable and array memory layout (default)

–nopad : disable changing variable and array memory layout

–fast : enable –O3 –ipo –static

–parallel : enable auto-parallelizer to generate multi-threaded code for loops that can safely execute in parallel

Advanced Optimizations …

- Inter-Procedural Optimization (IPO)
  - Application performance enhanced by:
    - Decreasing number of branches, jumps and calls
    - Reducing call overhead further by function inlining
    - Providing improved alias analysis leading to better vectorization and loop transformations
    - Enabling limited data layout optimization for better cache usage
    - Identifying memory references and reducing accesses
  - IPO works in two-phased automatic process
    - During the first phase, IPO creates an information file containing intermediate source code and optimization summary
    - In the second phase, compiler uses information file to achieve further optimizations such as function inlining etc.

Advanced Optimizations …

- Enable and Specify the Scope of the IPO by:

–ip : enable singe-file IPO (within files)

–ipo : enable multi-file IPO (between files)

–ipo_c: generate a multi-file object file (iop_out.o)

–ipo_S: generate a multi-file assembly file (iop_out.s)

- Modify the behavior of IPO by:

–ip_no_inlining : disable IPO inlining (need –ip / –ipo)

–ipo_obj : force generation of real object files (requires –ipo option)

Advanced Optimizations …

- Profile-Guided Optimization (PGO)
  - Works in three distinctive steps
  - Independent of other optimizations, but better with IPO
  - Bundles code based on frequency of usage
  - Minimizes no. of branches from the original build
  - Enables better branch prediction
  - Results in improved instruction cache usage
  - Benefits code with very large no. of branches, but few used at any particular time
  - Biggest gain from large and complex applications with many function calls and branches (especially with IPO)

Advanced Optimizations …

- How does PGO work ?
  - Compile for instrumentation using –prof_gen compiler flag
  - Run the instrumented code with a typical data
  - Dynamic information files (*.dyn and *.dpi) are created
  - Recompile the code with –prof_use compiler flag
  - Compiler creates an optimized code based on the information collected on the profile summary file
  - Choose –prof_file <fname> flag to specify summary file name
  - Use –prof_dir <dirname> flag to specify a directory for profiling output files (*.dyn and *.dpi)
  - Benefits depend on the application and data used
  - Can merge data files (using profmerge) to optimize code for more than one data sets

Parallel Computing Support

- Multi-Threading:
  - Using POSIX thread (pthread) libraries
  - With OpenMP directives
    - Compiler recognizes the industry standard OpenMP directives or pthread lib. calls and creates multi-threaded executables accordingly
    - Support available both in ecc (C, C++) and efc (Fortran) compilers
  - Auto-Parallelization
    - Use –parallel flag to enable auto-parallelization
    - Exploits loop level parallelism for multi-threaded execution
    - Very conservative in detecting loops for parallelization
    - Available both in ecc and efc compilers
- Multi-Processing:
  - Through variants of Message Passing Interface (MPI)
  - Can be used as parallel code in a single / distributed servers (cluster)
  - Works in a hybrid parallel mode with multiple threads/process

Optimization Report …

- Options to generate and manage optimization report (No optimization report by default)

–opt_report : generates an optimization report to stderr

–opt_report_file <fn> : sends to a file <fn> instead of stderr

–opt_report_level [level] : specifies the level of report verbosity (min | med | max)

–opt_report_routine <name> : reports on routines containing <name>

–opt_report_phase <phase> : specify the phase for which reports are generated for.

–opt_report_help : displays optimization phases available for reporting

optimization phases: ipo, ipo_inline, hlo, ilo, omp and all

Optimization Report

- Types of optimization reports generated
  - Report on individual modules
  - Information on pipelining
  - Prediction and code emission details
  - Register allocation reporting
  - Detailed information on inlining
  - Static and dynamic branch counts
  - Loop blocking, loop unroll and jam report
  - Instruction and bundle count (static and dynamic)
  - Execution time estimates in cycles

Libraries

- Large number of libraries, both archived and shared
- Useful for portability, functionality and performance
- Static libraries linked at link time and shared libraries linked at run time as dynamically-shared objects
- List of some key libraries and linker flags:
  - libF90.a, libimf.a, libm.a libintrins.a, libc.a, libcprts.a …
  - Use –static to prevent linking with shared libraries
  - Use –shared to produce a shared object
  - Use –posixlib option to invoke POSIX bindings library
  - Use –Vaxlib flag to link with portability library
  - Use –C90 to link with alternate I/O for mixed output with C
  - Use –nostdlib NOT to link with standard libs and startup files

Intel Math Kernel Library (MKL)

- Highly optimized library modules for mathematical, scientific, engineering and financial applications
- Support for multi-processing and multi-threading
- Greatly enhances application performance and reduces developmental cost
- Cluster and ScaLAPACK support for Linux
- Key components of MKL include:
  - Linear Algebra (LAPACK) and BLAS (Levels 1,2 and 3)
  - Discrete Fourier Transforms (DFT)
  - Vector Statistical Library functions (VSL)
  - Vector transcendental Math Lib functions (VML)

Intel Integrated Performance Primitives (IPP)

- Library of signal and image processing, multi-media and vector math functions
- Common APIs across platforms and operating systems
- Non-assembly code for boosting application performance
- New features for Cryptography and Signal Processing
- Enhanced support for Audio, Video, Speech coding and Image processing
- Achieve major performance gain with minor code changes

Tools …

- Profiling Tools
  - *Vtune* from Intel for sampling and call graphing
  - *Prospect* from HP for process and kernel profiling
  - *pdp* from HP, a library and curses tool for user profiles
  - *oprofile* from Sourceforge for system profiling
  - *gprof* / *prof* from opensource, available in all distributions
  - *cprof* from Corel for multi-threaded C and C++ applications
  - *pfmon* from HP for collecting performance metrics
- Tracing Tools
  - *LTT* from Opensys for collecting traces of system calls
  - *strace* and *ltrace* from Rpmfind, online tracing tools for library and system calls

    (similar to *truss* in Solaris and *tusc* in HP-UX)

Tools

- System Monitoring Tools
  - *Powertweak* from Sourceforge to tune performance settings
  - *Truespeed* for CPU load monitoring
  - *top*/*gtop*/*ktop* for processes and CPU load monitoring
  - *Vmstat* for monitoring system level virtual memory usage
- Debugging Tools
  - *gdb*, GNU debugger from Opensource and *Idb* from Intel
  - *tcpdump* for memory and malloc debugging
  - *Dprobes* from IBM for kernel debugging
- APIs and Frameworks
  - *IPP*, Intel Integrated Performance Primitives for signal and image processing, multimedia and vector math functions
  - *PCP*/*PMAPI* from SGI for collecting performance metrics

VTune Performance Analyzer …

- Collects performance data while application runs
- Works in native and remote data collection modes
- Ability to analyze from system level, source or even instruction level
- Available for Windows (GUI + command line) and Linux (Red Hat, SuSE – command line only)
- Support for up to 64 CPUs
- Java (32-bit) sampling and call graph functionality
- APIs for Pause and Resume capabilities

VTune – Key Features …

- Sampling
  - Uses sampling collector to periodically interrupt the processor to collect data, either time-based or event based
  - No need to modify source
  - Sampling is system-wide
  - Sampling overhead is minimal and can be controlled through user interface
- Call Graphing
  - Profiling of code only in application level
  - Tracks function entry and exit points of a code
  - Uses binary instrumentation
  - Data used for hotspots, program flow and critical functions and call sequences

Livermore Fortran Kernels (LFK)

OS / Compiler Flags	Max. Rate (MFLOPS)	Geo. Mean (MFLOPS)	Min. Rate (MFLOPS)
Linux / -O0	49.6	18.4	7.6
Linux / -O2	2039.7	353.3	48.8
Linux / -O3	3495.5	384.6	60.0
Linux /-O3 -ipo	3489.2	387.8	60.6
HP-UX / +O3	3337.8	502.2	78.5

Linux: RX2600, 2 CPU, 900 MHz, Debian 2.4.18, 8 GB Mem.
HP-UX: RX2600, 2 CPU, 900 MHz, HP-UX 11.22, 12 GB Mem.

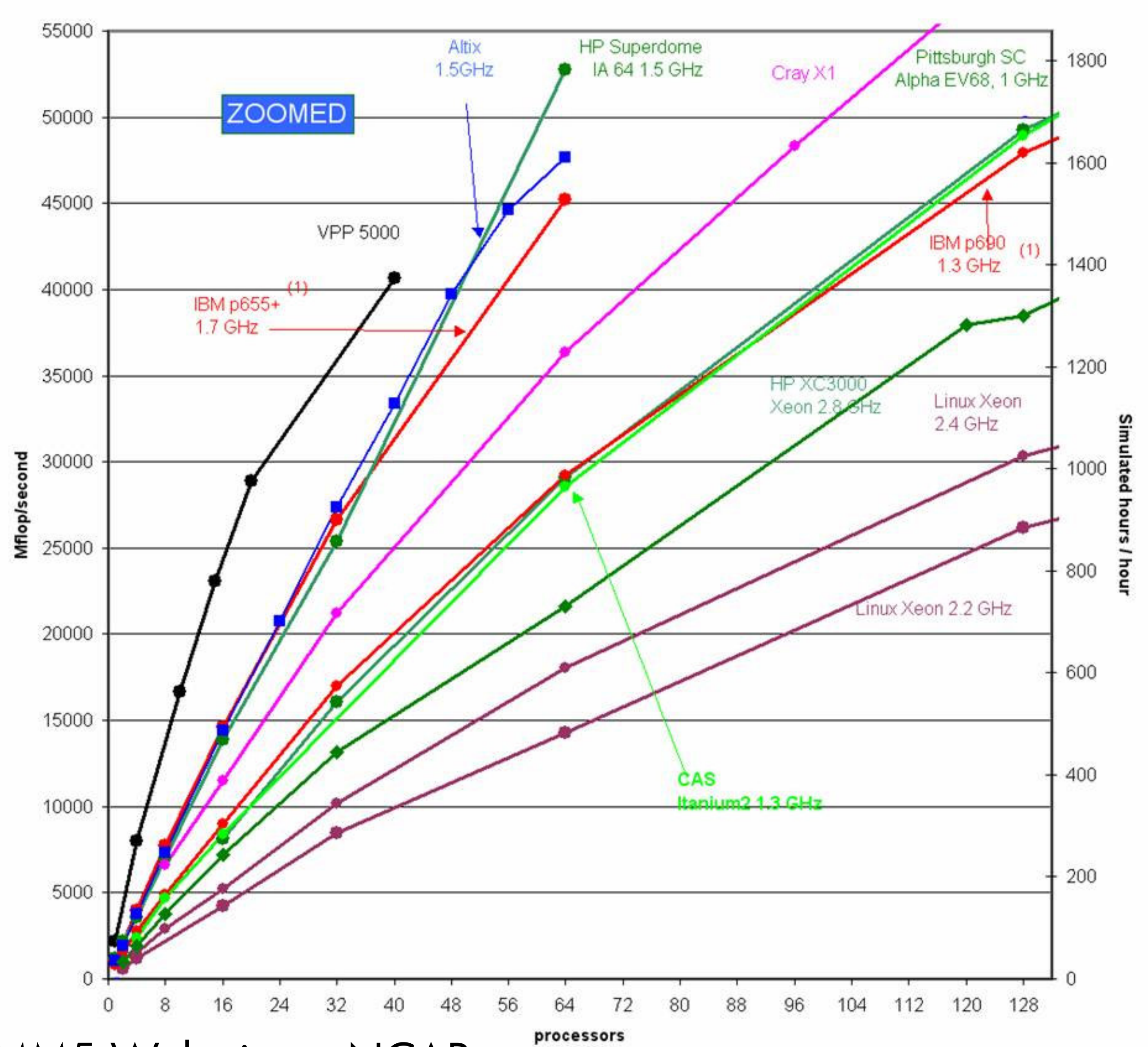
Performance of a CFD Code

OS / Compiler Flags	Elapsed Time (seconds)
Linux / -O0	27,172
Linux / -O1	5,978
Linux / -O2	3,143
Linux / -O3	3,090
Linux / -O3 -ipo	3,145
Linux / -O3 -pgo	3,071
HP-UX / +O3 (default page)	3,252
HP-UX / +O3 (64 MB page)	1,604

Linux: RX2600, 2 CPU, 900 MHz, Debian 2.4.18, 8 GB Mem.

HP-UX: RX2600, 2 CPU, 900 MHz, HP-UX 11.22, 12 GB Mem.

MM5 Results: Superdome is Leader



Source: MM5 Web site at NCAR

MM5 Results: A Comparison

No. of CPUs	HP Superdome (1.5 GHz)	SGI Altix (1.5 GHz)	IBM (p655+) (1.7 GHz)
1	1,218	1,032	1,704
2	2,188	1,901	2,098
4	3,582	3,735	3,970
8	7,000	7,354	7,710
16	13,872	14,389	14,620
32	25,388	27,377	26,641
64	52,782	47,678	45,239

Source: MM5 web site at NCAR

MM5 Results from RX2600 Cluster

No. of CPUs	HP-UX (Infiniband)	Linux (XC6000) (Quadrics)
1	1,596	1,071
2	1,850	1,985
4	3,453	3,576
8	6,582	7,170
16	12,071	14,458
32	25,418	26,917
64	43,890	46,501

Note: XC6000 Results contributed by Enda O'Brien

Results from Weather Research Forecasting (WRF) Application

No. of CPUs	Elapsed Time (sec)	Speedup	% Parallel Efficiency
1	10,028	1.00	---
4	3,004	3.34	83.5
8	1,565	6.41	80.1
16	882	11.37	71.1
32	520	19.28	60.3

System: XC6000 (RX2600, 1.5 GHz, Linux, Quadrics)

Data: CONUS, Grid 12 km, Step 72 s, 6 hr Simulation

WRF Comparison from RX2600 Cluster

No. of CPUs	HP-UX (Infiniband)	Linux (XC6000) (Quadrics)
16	1:52:29	1:50:20
20	1:36:45	1:33:57
32	1:02:29	1:01:39
64	37:02	38:25
128	21:21	21:48

Data: Custom, Grid 3 km, Step 15 s, 3 hr Simulation

Note: Times are in hh:mm:ss format

Comparison of a Signal Processing Code

No. of CPUS	HP-UX (Hyperfabric)	Linux (XC6000) (Quadrics)
8	80.18	87.16
16	52.04	54.53
32	37.09	33.48

Note: Times are in seconds

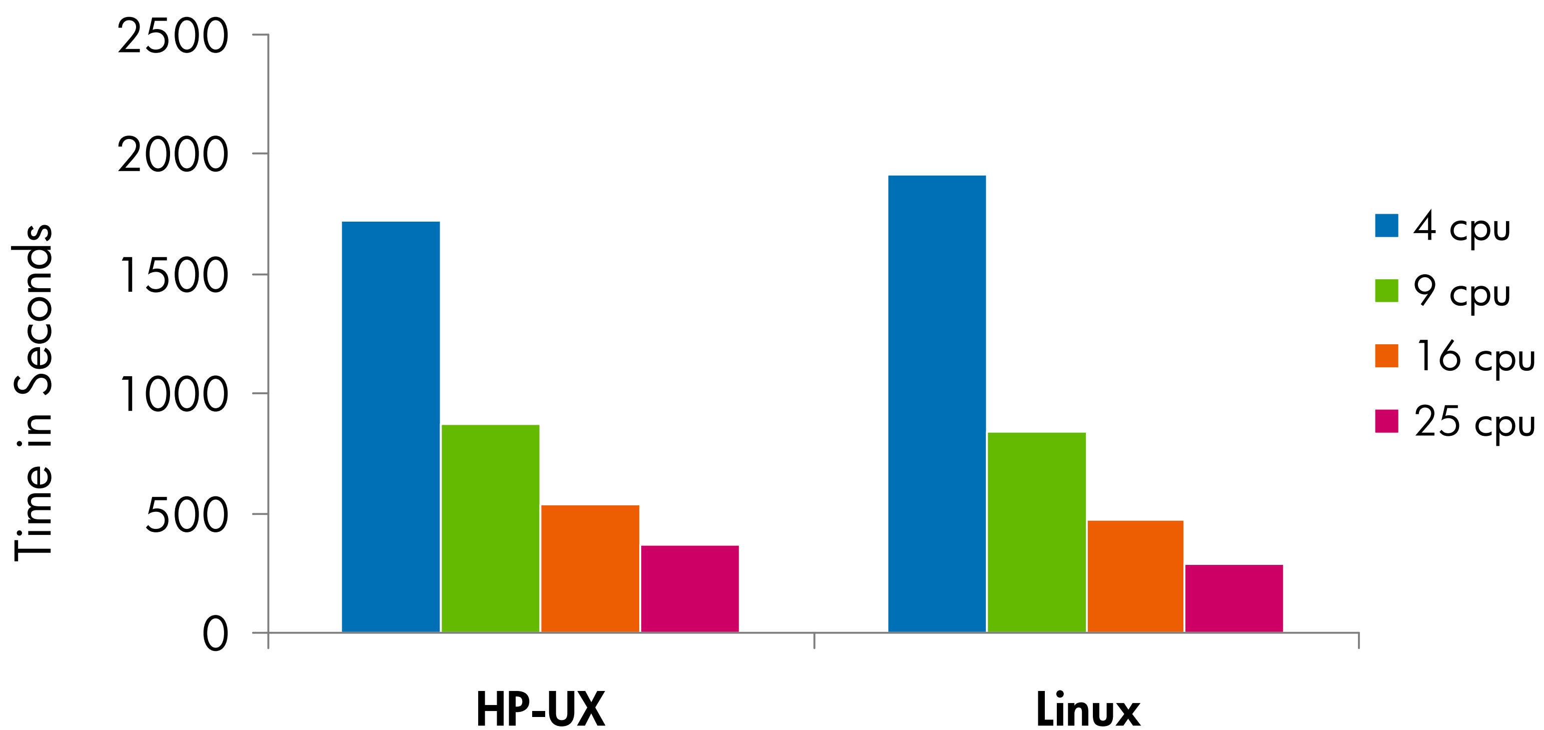
Comparison of a Search Algorithm

No. of CPUS	HP-UX (Hyperfabric)	Linux (XC6000) (Quadrics)
2	1849.23	2576.23
4	913.91	1177.37
8	461.02	593.77
16	268.83	378.91
32	142.90	185.96

Note: Times are in seconds

NASPAR (Class: C) on RX2600 Cluster (Interconnect: Myrinet)

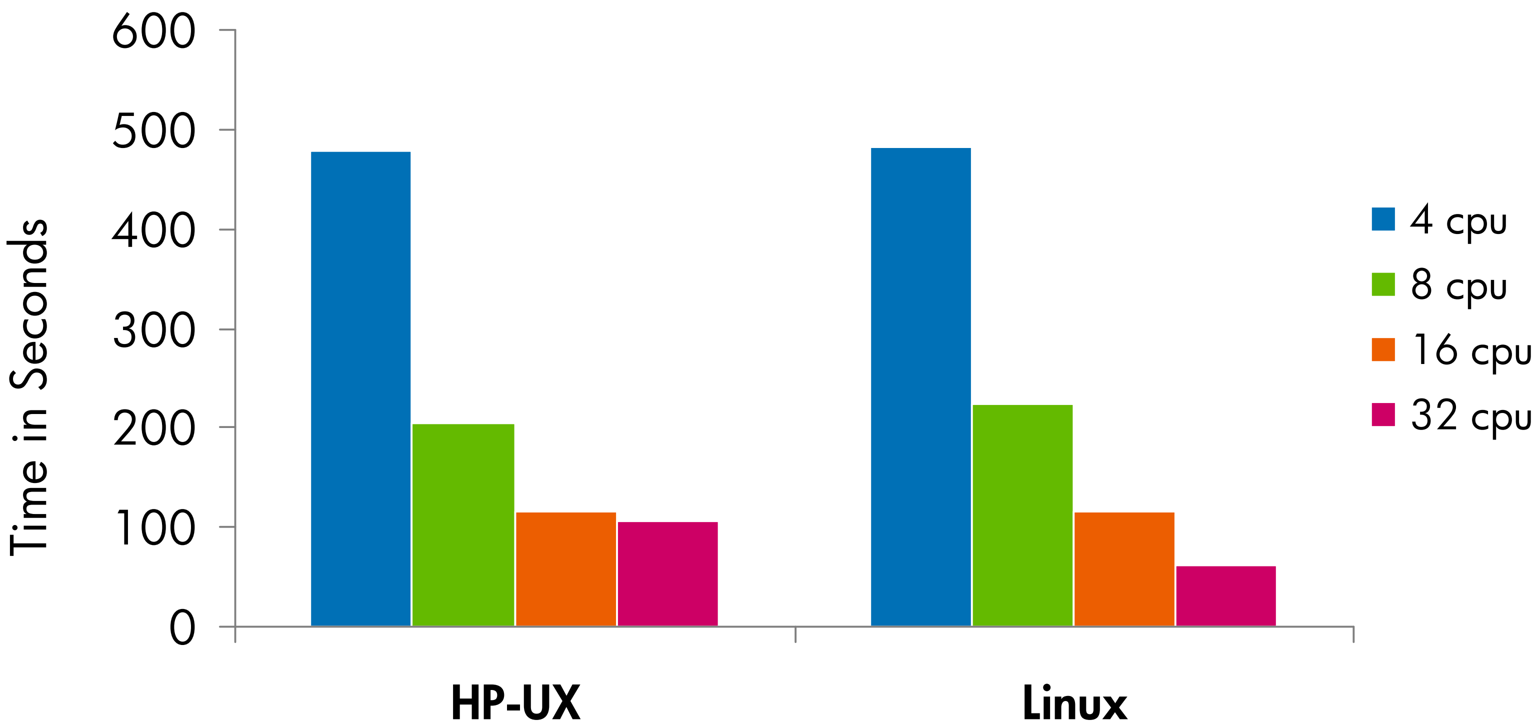
Kernel: BT



SMALLER VALUE IS BETTER

NASPAR (Class: C) on RX2600 Cluster (Interconnect: Myrinet)

Kernel: LU



SMALLER VALUE IS BETTER

HP WORLD 2004 Solutions and Technology Conference & Expo

Hardware and Software Comparison of a Pthread based C Code

Hardware and Software	Elapsed Time (sec)	CPU time (sec)	% Parallel Efficiency
RX2600, 1.5 GHz, Linux, ecc 7.1	37.8	72.0	190
RX2600, 1.5 GHz, Linux, ecc 8.0	35.2	68.9	195
RX2600, 1.5 GHz, HP-UX 11.23, cc (hp)	27.8	51.6	186
DL140, 3.2 GHz, Linux, icc 8.0	32.2	61.2	190

Conclusions

- Easy to migrate to Linux
- Intel Compilers can do the job
- Intel compilers are improving fast on functionality and performance
- Advanced tuning and optimization report are excellent
- Tools and Libraries are adequate but more will be needed
- Performances are comparable to HP-UX based systems
- Variable pages and means to build 32-bit addressable codes will be of great help

HP WORLD 2004
Solutions and Technology Conference & Expo

Co-produced by:

interex
shared knowledge • shared power

encompass
AN HP USER GROUP

RECOMMENDED TRAINING VENUE FOR THE
HP Certified Professional