# 基于 NAS Benchmarks 的 ORC 性能测试*)

林海波　汤志忠
(清华大学计算机科学与技术系　北京100084)

## The Performance of ORC with NAS Benchmarks

LIN Hai-Bo　TANG Zhi-Zhong
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)
E-mail: linhaibo99@mails.tsinghua.edu.cn　http://www.tsinghua.edu.cn

**Abstract**　Itanium is the first generation product processor based on IA-64 architecture. ORC(Open Research Compiler) provides an open source IPF(Itanium Processor Family) research compiler infrastructure. We have compiled and run NAS Benchmarks on the Itanium machine. This paper briefly describes the performance of orcc, sgicc and gcc in the following 3 ways: execution time, compilation time, and executable file size. The results show that orcc has nearly the same performance as sgicc, which is 2 fold faster over gcc in the aspect of execution time. We also find that even with the best-optimized program, the utilization ratio of process resources is no more that 70%.

**Keywords**　Itanium, ORC, NAS benchmarks, Performance testing

## 1　引言

安腾(Itanium)[1]处理器是HP/Intel公司推出的第一代基于IA-64体系结构的处理器。IA-64体系结构是一种64位的支持显式指令级并行计算(Explicit Parallel Instruction Computing, EPIC)的体系结构，它实现了一系列新特性，支持开发更大的指令级并行性(Instruction Level Parallelism, ILP)，突破了传统体系结构的性能限制。这些新特性包括：猜测执行(Speculation)、条件执行(Predication)、大量的寄存器、寄存器堆栈及先进的分支结构等。这种EPIC体系结构使得编译器可以和硬件传递更多的信息，以软硬件配合、静态预测与动态执行相结合的方式更好地提高程序执行的性能。

ORC(Open Research Compiler)[2]是Intel公司提供的基于IA-64体系结构的开放源码的研究型编译器，其原型是SGI公司的Pro64编译器。它们针对IA-64体系结构的特点做了专门的优化，以充分利用安腾处理器的硬件支持。本文将以NAS Benchmark为例，对ORC、Pro64(version 0.13)以及GNU gcc的性能做一比较，并进一步分析性能提高的原因和潜力。

## 2　安腾处理器

安腾处理器是HP/Intel公司IA-64体系结构的第一个硬件实现。为了进一步提高系统的性能，IA-64体系结构包含了以下的新特性[1]：

(1)显式并行性

·提供了编译器和处理器之间的协同机制；

·提供了大量的硬件资源以开发指令级并行性；

·各128个整数/浮点寄存器、64个条件(Predicate)寄存器和8个分支寄存器；

·支持多个功能部件(Itanium中包括2个整数功能部件、2个浮点功能部件、2个分支功能部件和3个分支功能部件)。

(2)对指令级并行性的支持

·猜测执行(Speculation)：减小存储器访问延迟；

·条件执行(Predication)：消除分支；

·软件流水(Software Pipelining)；

·分支预测(Branch Prediction)；

(3)对软件性能的支持

·对软件模块化的特殊支持；

·高性能的浮点体系结构；

·特殊的多媒体指令。

指令级并行性(Instruction Level Parallelism, ILP)是指在同一时刻并行执行多条指令。IA-64体系结构允许指令束(Bundle)中不相关的指令(1个指令束中包含3条指令)并行执行，并可以在一个时钟周期发射多个指令束(在当前的Itanium处理器中每周期可发射2个指令束)。编译器可以合理地调度指令的执行顺序或并发的线程，以充分利用寄存器和功能部件等硬件资源。

IA-64体系结构提供了编译器和处理器之间协同工作的机制，如指令模板(Instruction Template)、分支暗示(Branch Hint)、cache暗示等，这使得编译器可以将编译时信息传递给处理器。这些信息可以有效地减小由于条件预测失败或cache失效而带来的性能损失。如IA-64中的LOAD/STORE操作带有1个2位的cache hint域，编译器根据时间/空间局部性来填写这个域，在运行时处理器根据cache hint域的信息来确定数据在cache体系结构中的位置。

IA-64支持两种形式的猜测执行：控制猜测(Control Speculation)和数据猜测(Data Speculation)。控制猜测是指将条件分支语句之后的操作移至条件分支之前；数据猜测是指将Store操作之后的Load操作移至Store操作之前。只有当

*)基金项目：国家自然科学基金资助项目(60173010)。林海波　博士生，主要研究领域为体系结构，编译技术。汤志忠　教授，博士生导师，主要研究领域为计算机并行算法，并行编译技术。

编译器认为猜测执行可以提高性能的时候才进行猜测，这就必须保证：(1)猜测代码的执行概率较高且恢复代码较少；(2)猜测执行可以提供更多的ILP。猜测执行是编译器根据统计信息来提高ILP的主要方法之一。

IA-64提供了条件执行机制，IA-64可以根据每条指令所对应的条件寄存器的值来决定这条指令是否执行。如果条件为真，则该指令执行；否则该指令被当作空操作。条件执行可以去除指令中的分支操作，把控制相关转换成数据相关，以形成更大的基本块(basic block)，有利于指令调度。

IA-64可以通过编译器控制的重命名机制来避免过程调用时不必要的寄存器溢出(Spilling)和填充(Filling)。当进行过程调用时，被调用过程通过alloc指令来指明它所需的寄存器帧(Frame)的大小，访问寄存时将通过一个寄存器基址将指令中的虚拟寄存器映射到物理寄存器中。如果寄存器不足，调用过程的一部分寄存器将被溢出到内存，直到有足够的寄存器供被调用过程使用为止。在过程返回时，寄存器基址也将被恢复，使得原先的过程可以访问过程调用前的寄存器。如果调用过程有寄存器被溢出到内存中，则处理器将阻塞，直到它们被恢复。

除了通过条件执行来去除分支，IA-64也提供了其他的分支预测机制。分支预测指令可以用来传递目标地址信息和分支操作的位置，编译器将指明应该采用静态预测还是动态预测，处理器可以根据这些信息来初始化分支预测结构，这样即使是第一次执行分支操作也可以正确地预测分支方向。对于间接地址跳转，分支寄存器被用来保存分支的目标地址。分支预测指令将指明使用哪一个分支寄存器，这样就可以提前得到分支的目标地址。对于简单的确定次数的循环，分支预测指令可以完全正确地预测循环的结束。

软件流水可以使一个循环的不同循环体并行执行，然而这些同时执行的不同循环体需要访问不同的寄存器。IA-64提供了寄存器旋转机制(Rotating Register)，使得无需循环展开就可以实现自动的寄存器重命名。

IA-64支持IEEE所定义的单精度、双精度及扩展双精度浮点数据类型。Itanium中共有128个浮点寄存器，其中的96个可以用作旋转寄存器。另外还有多个浮点状态寄存器用来支持猜测执行。

IA-64的多媒体指令可以将64位的通用寄存器看作8个8位、4个16位或2个32位数据的连接，这些指令可以并行地处理每段数据，并保持与MMX和SIMD指令的兼容。

## 3 ORC(Open Research Compiler)

ORC是Intel公司提供的基于IA-64体系结构的开放源码的研究型编译器，它的前身是SGI公司的Pro64编译器。

ORC旨在提供一个编译器的开放性研究平台及基础架构，它现已支持以下几个新特性[2]：ORC采用了基于区域(region-based)的编译技术，通过生成合适大小和形状的编译单元来进行优化，并可控制编译过程的时间/空间复杂度。ORC还提供了丰富的profiling支持，如edge-profiling、value-profiling等。

ORC针对IA-64体系结构做了专门的优化。为了支持条件执行，ORC程序进行了IF条件转换(if-conversion)，并创建了可重构的条件位关系数据库(PRDB)，以供分析和查询各条件位之间的关系。通过IA-64的并行比较指令，ORC可将部分类型的串行比较转换成并行比较，从而减小控制高度。ORC的全局指令调度基于SEME region，通过查询PRDB，根据路径的执行概率构造DAG，然后与资源管理相配合完成指令调度。ORC实现了控制/数据猜测执行，以及相应的恢复代码生成。最后，ORC还实现了一个参数化的机器模型和进行资源管理的微调度器(micro-scheduler)，通过微调度器和指令调度相配合，可以使生成的代码更合理地利用处理器资源。灵活的机器模型可以使ORC较为容易地支持下一代安腾处理器。

## 4 NPB 2-serial Benchmarks

NASA的研究人员开发了一套代表流体动力学计算的应用程序集，它已经成为超级计算机性能测试的标准测试程序之一[3]。NAS Benchmarks共包含8个程序，其中有5个核心(kernel)benchmark相对较小，但也足以对当今超级计算机的处理器、存储子系统以及通信结构进行性能测试。这5个核心程序是：CG(Conjugate Gradient)、EP(Embarassingly Parallel)、FT(Fourier Transform)、IS(Integer Sort)和MG(MultiGrid)[4,5]。

NAS Parallel Benchmarks(NPB)有几个版本。NPB 1是一个相当简单的版本，它包括几个简单的FORTRAN程序，只是能够计算出所需的结果。

NPB 2是一个并行版本，它主要用FORTRAN 77语言和MPI库编写。它的设计初衷就是无需(或很少的)改动就能够在绝大多数的并行计算机上运行。由于我们进行测试的是单处理机的HP i2000工作站，无需处理机间的通信，因此NPB 2并不是特别适合我们的测试。

NPB 2-serial是NPB 2的单处理机版本，NPB 2中的并行化部份被全部去除，因此它可以作为工作站的性能测试程序集。我们这里采用的是NPB 2.3-serail，通过运行由ORC(orcc)、Pro64(sgicc)和gcc编译所得到的二进制程序，来对这3个编译器进行初步的性能评价[6]。

NAS为每个测试程序的输入集定义了5个不同级别的规模：S(ample)、W(orkstation)、A、B和C，以适应不同计算机系统的需要。这里我们采用的是为工作站设计的W级输入集。

## 5 测试方法

我们将分别用orcc、sgicc和gcc编译NAS Benchmark，运行环境为单CPU的HP workstation i2000工作站。HP i2000工作站的配置如表1所示。

表1 HP workstation i2000的配置

| 处理器 | 733MHz Itanium |
|---|---|
| 总线频率 | 133MHz |
| 高速缓存(cache) | 16K(L1)，64K(L2)，2M(L3) |
| 内存 | 1G |

编译选项为各编译器自定义的O0(无优化)、O1(最小优化)、O2(常用优化)和O3(最大优化)。在orcc/sgicc中，绝大部分优化选项在O2中被打开，如全局指令调度、IF转换、向量相关性分析、软件流水、全局寄存器分配、循环展开等，O3选项除包含O2所定义的全部优化选项外，还包括了多重循环优化(Loop Nest Opt.，LNO)。NAS Benchmark中的RAND变量取randi8_safe，此时orcc和sgicc可成功编译全部的benchmark，gcc编译FT程序时失败。

## 6 实验结果

### 6.1 执行时间

首先来比较经过3个编译器编译后的应用程序性能。表2给出了采用-O3编译选项时，各个程序用不同的编译器编译后的执行时间。

表2 NAS Benchmark 的执行时间(-O3)

| Benchmark / Compiler | BT | CG | EP | FT | IS | LU | MG | SP |
|---|---|---|---|---|---|---|---|---|
| | fp | fp | int | fp | int | fp | fp | fp |
| orcc | 68.22 | 7.57 | 9.44 | 1.73 | 0.6 | 153.28 | 4.53 | 155.62 |
| sgicc | 71.12 | 7.01 | 9.47 | 1.71 | 0.58 | 143.37 | 4.4 | 152.46 |
| gcc | 139.07 | 16.13 | 22.26 | --- | 1.02 | 284.01 | 12.3 | 333.54 |

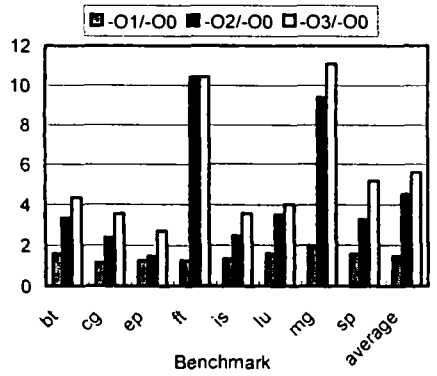


图1 orcc 优化编译选项的加速比

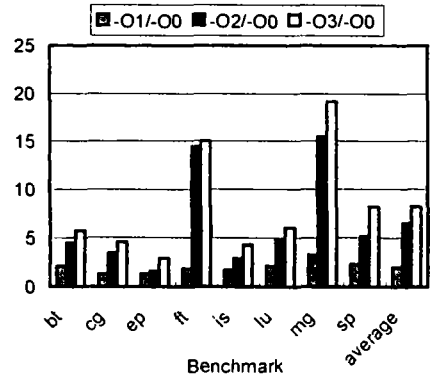


图2 sgicc 优化编译选项的加速比

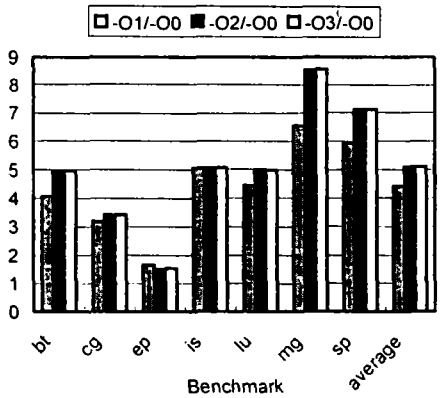


图3 gcc 优化编译选项的加速比

图1～图3分别是 orcc、sgicc 和 gcc 所用不同编译选项所得的加速比。由图可知，orcc 的优化技术所得到的平均加速比约为5.6(-O3/-O0)，sgicc 所得到的平均加速比约为8.2，这主要是因为采用 O0选项时 orcc 编译所得的程序执行速度比 sgicc 快。其中 FT 和 MG 两个程序的加速比最大，分别为10(orcc)/15(sgicc)和11(orcc)/19(sgicc)，这是由应用程序本身的特性所决定的。图4给出了 orcc 与 sgicc/gcc 的加速比。由于 FT 没有能够通过 gcc 的编译，因此图中只比较了7个 Benchmar 的数据。由图中可以看出，orcc 和 sgicc 的性能大致相同，只有 BT 和 EP 两个程序的执行时间略少于 sgicc，其余皆大于 sgicc。Orcc/sgicc 的平均加速比为0.97，说明 sgicc 的性能略强。与 gcc 相比，orcc 和 sgicc 都显示出明显的性能优势，orcc/gcc 的平均加速比为2.13。

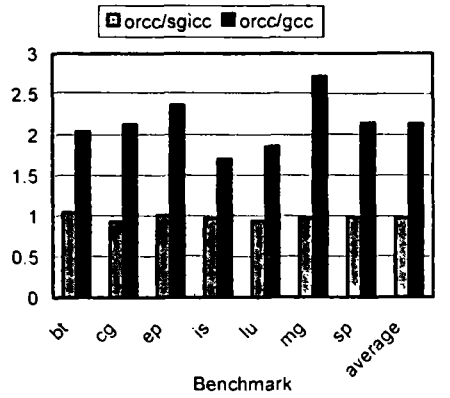


图4 orcc 与 sgicc/gcc 的性能比较(-O3)

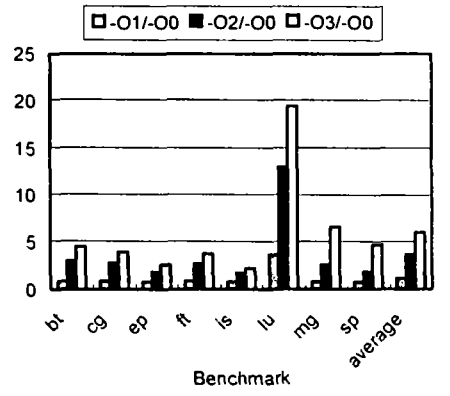


图5 orcc 的编译时间

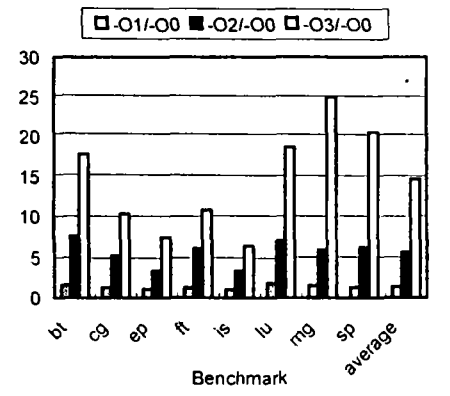


图6 sgicc 的编译时间

### 6.2 编译时间

图5～图7所示为 orcc、sgicc 和 gcc 采用不同编译选项的编译时间比较。图8所示为采用-O3选项时 orcc/sgicc 的编译时间与 gcc 之比。由图可知，对于 orcc 来讲，优化编译技术使

编译时间平均增加了约5倍，而 sgicc 则增加了约14倍。可见 orcc 的基于区域的编译技术有效地控制了编译过程的时间复杂度（约为 sgicc 的37%），并且基本保持了编译所得代码的性能（只比 sgicc 低3%）。Gcc 的编译时间具有绝对的优势，约为 orcc 的10%和 sgicc 的3%，而 gcc 的优化选项并没有引起编译时间太大的变化，并且当采用-O1优化选项时，编译时间没有增加，反而比采用-O0选项时减少了22%。

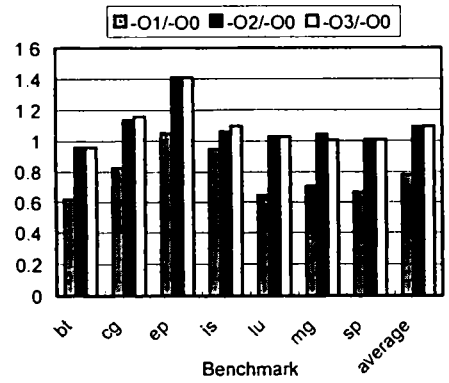


图7 gcc 的编译时间

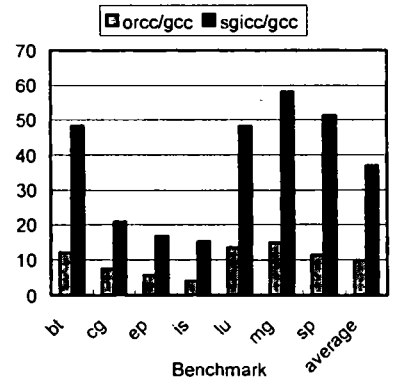


图8 orcc/sgicc 与 gcc 的编译时间比较

### 6.3 文件大小

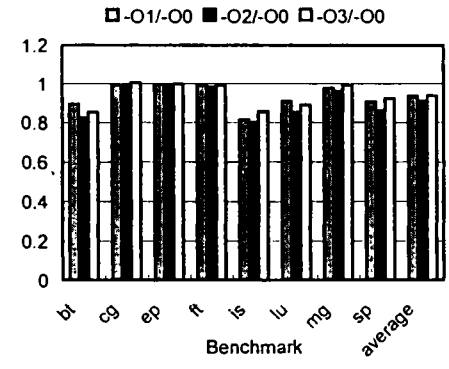


图9 orcc 编译的文件大小

图9～图11所示为 orcc、sgicc 和 gcc 采用不同编译选项时编译所得二进制文件的大小比较，图12所示为采用-O3选项时 orcc/sgicc 和 gcc 编译所得二进制文件大小之比。由图可知，orcc、sgicc 和 gcc 的优化编译技术都使得目标文件的大小有所减小，这是由于优化编译删除了冗余操作，减少了指令条数。Orcc 和 sgicc 在目标文件大小方面没有太大的差别，而 gcc 编译所得的文件仅为 orcc/sgicc 编译所得文件大小的1/7。由于 orcc/sgicc 在目标代码中附带了一些编译信息以供处理器参考，这可能是导致可执行文件变大的一部分原因。

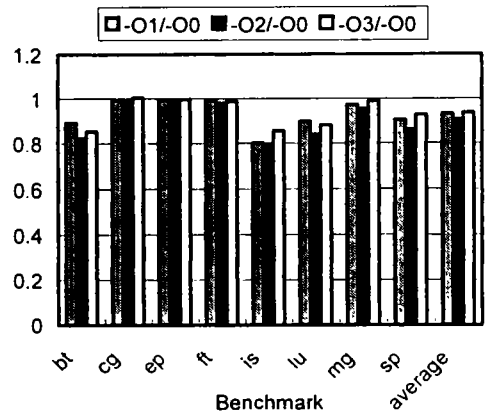


图10 sgicc 编译的文件大小

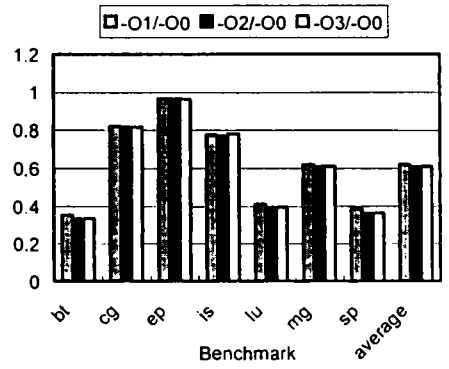


图11 gcc 编译的文件大小

### 6.4 采样分析

pfmon[7]是 HP 公司开发的一个性能监测工具，它通过访问安腾处理器中提供的性能监测部件（PMU）来对应用程序进行计数和采样，从而获得程序运行的动态统计信息。图13对比了采用 orcc 的-O0和-O3选项编译后应用程序各类操作的变化，图14对比了时钟周期分布的变化。

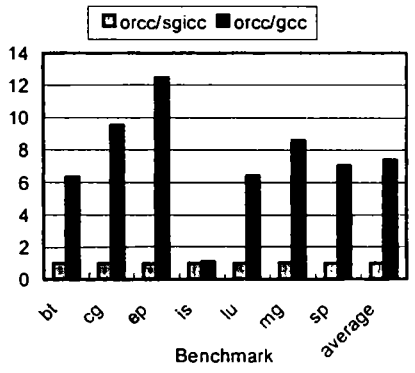


图12 orcc 与 sgicc/gcc 编译的文件大小比较

由前面的测试结果可知，FT 和 CG 是加速比最高的两个程序，图14也表明了这一点，但 pfmon 得到的加速比（约为6）没有像图1中显示的那样高（约为10）。由图中可以看出，经过优化编译后，MG 的指令条数减少为原来的1/7，其中空操作和取数操作（LOAD）也有相应比例的减少，因此取指周期（INST_FETCH_CYCLE）和存储器访问周期（MEMORY_CYCLE）有了较大幅度的下降，这与图14所示的加速比（6）基

本吻合。FT的指令条数虽然没有减少得那么多，但其取数操作却减少为原来的1/8，由此带来了指令执行时延(EXCUTION LATENCY_CYCLE)(1/26)、取指周期(INST_FETCH_CYCLE)(1/28)和存储器访问周期(MEMORY_CYCLE)(1/6)的显著减少，可见LOAD操作的优化对程序的加速比贡献很大。

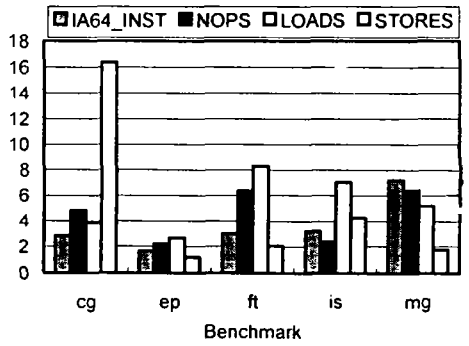


图13 orcc编译的指令条数比较(O0/O3)

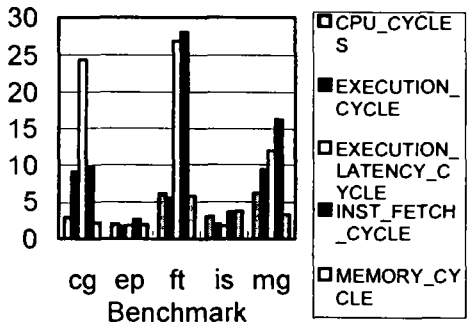


图14 周期分布(O0/O3)

从统计结果(表3)中还可以看出，尽管经过强劲的优化措施，程序执行过程中的空操作仍占有相当的比例(36%)，这说明处理器资源还有30%以上的空闲，如果能有效地利用这些处理资源，将获得更高的加速比或吞吐量。

表3 各类操作所占总操作的百分比

| 操作类型 \ 编译选项 | -O0 | -O3 |
|---|---|---|
| 空操作(NOP) | 45% | 36% |
| 取数操作(LOAD) | 22% | 15% |
| 存数操作(STORE) | 5% | 6% |

**结论** 本文通过在安腾处理器上编译运行NAS Benchmarks，从程序的运行时间、编译时间和产生的可执行文件的大小3个方面，评价了orcc、sgicc和gcc的性能。通过上文的比较，大致可得出如下结论：

·orcc和sgicc性能相当，约为gcc的2倍。

·orcc比sgicc有效地控制了编译时间，但仍比gcc慢10倍。

·orcc和sgicc产生的可执行文件大小相当，约为gcc产生文件的7倍。

通过对程序运行时的监测，发现空操作占有相当的比例，处理器还有30%的空闲。减少存储器访问操作或隐藏存储器访问时延可以显著地提高程序的性能。

## 参考文献


1 Intel IA-64 Architecture Software Developer's Manual. Volume 1: IA-64 Application Architecture. 2000

2 Roy J, Sun C, Chengyong W. Open Research Compiler(ORC)for Itanium Processor Family(IPF). MICRO-34 Tutorial. 2001

3 http://science.nas.nasa.gov/Software/NPB

4 Agarwal R C, Alpern B, Carter L, Gustavson F G, Klepacki D, Lawrence R, Zubair M. High PerformanceParallel Implementations of the NAS Kernel Benchmarks on the IBM SP2. IBM Systems Journal, 1995, 34:263～272

5 Goedecker S. Fast Radix 2,3,4, and 5 Kernels for Fast Fourier Transformations on Computers with overlapping multiply-add instructions. SIAM Journal on Scientific Computing, 1997, 16:1605～1611

6 Boisseau J, Carter L, Gatlin K, Majumdar A, Snavely A. NAS Benchmarks on the Tera MTA. In: Workshop on Multi-Threaded Execution, Architecture, and Compilers. 1998

7 ftp://ftp.hpl.hp.com/pub/linux-ia64/pfmon-0.06.tar.gz


(上接第34页)

最终假设 $h_f$ 的均方误差 $\varepsilon=E_{i\sim D}[(h_f(x_i)-y_i)^2]$ 的上边界为：$\varepsilon\leqslant 2^T\prod_{t=1}^{T}\sqrt{\varepsilon_t(1-\varepsilon)}$。

## 参考文献


1 Valiant L G. A theory of the learnable. Communications of the ACM 1984,27(22):1134～1142

2 Kearns M K, Vazirani L G. Learning Boolean formulae or finite automata is as hard as factoring:[Technical Report TR-14-88]. Harvard University Aiken Computation Laboratory, Aug. 1988

3 Kearns M J, Vazirani L G. Cryptographic limitations on learning Boolean formulae and finite automata. Journal of the Association for Computing Machinery, 1994,41((1):67～95

4 Schapire R E. The strength of weak learnability. Machine Learning, 1990,5(2):197～227

5 Freund Y. Iyer R, Schapire R E, Singer Y. An efficient bossting algorithm for combining preferences. In: Machine Learning: Proc. of the Fifteenth Int Conf. 1998

6 Freund Y, Schapire R E. Adecision-theoretic generalization of online learning and an application to boosting. Journal of Computer and System Science, 1997,55(1):119～139

7 Dietterich T G, Bakiri C. Solving multiclass learning problems viaerror-correcting output codes. Journal of Artificial Intelligence Research, 1995,2:263～286

8 Schapire R E, Singer Y. Using output codes to boost multiclass learning problems. In: Machine Learning: Proc. of the Fourteenth International Conference, 1997. 313～321

9 Schapire P E, Singer Y. Improved boosting algorithms using confidence-related predictions. In: Proc. of the Eleventh Annual Conf. on Computational Learning Theory, 1998. 80～91

10 Friedman J, Hastie T, Tibshirani R. Additive logistic regression: a statistical view of boosting: [Technical Report]. 1998

11 Schapire R E, Singer Y. BoostTexer: A system for multiclass multi-label text categorization. Machine Learning, 1998

12 刁力力，等. 数据挖掘与组合学习. 计算机科学，2001，28(7)

13 涂承胜，刁力力，鲁明明，陆玉昌. Boosting家族Boost-by-majority系列代表. 计算机科学，2003，30(4)