

面向异构多核处理器的并行代价模型

黄品丰^{1,2*}, 赵荣彩^{1,2}, 姚远^{1,2}, 赵捷^{1,2}

(1. 信息工程大学, 郑州 450002; 2. 数字工程与先进计算国家重点实验室, 郑州 450002)

(* 通信作者电子邮箱 huangpinfeng007@sina.com)

摘要: 现有的并行代价模型大多是面向共享存储或分布存储结构设计的, 不完全适合异构多核处理器。为解决这个问题, 提出了面向异构多核处理器的并行代价模型, 通过定量刻画计算核心运算能力、存储访问延迟和数据传输开销对循环并行执行时间的影响, 提高加速并行循环识别的准确性。实验结果表明, 提出的并行代价模型能有效识别加速并行循环, 将其识别结果作为后端生成并行代码的依据, 可有效提高并行程序在异构多核处理器上的性能。

关键词: 自动并行化; 并行代价模型; 异构多核; 数据传输开销; 加速并行循环

中图分类号: TP314 **文献标志码:** A

Parallel cost model for heterogeneous multi-core processors

HUANG Pinfeng^{1,2*}, ZHAO Rongcai^{1,2}, YAO Yuan^{1,2}, ZHAO Jie^{1,2}

(1. Information Engineering University, Zhengzhou Henan 450002, China;

2. State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou Henan 450002, China)

Abstract: The existing parallel cost models are mostly devised for shared memory or distributed memory architecture, thus not suitable for heterogeneous multi-core processors. In order to solve the problem, a new parallel cost model for heterogeneous multi-cores was proposed. It described the impact of computing capacity, memory access delay and data transfer cost on parallel execution time of loops quantitatively, thus improving the veracity of accelerated parallel loop recognition. The experimental results show that the proposed model can effectively recognize the accelerated parallel loops. Using its recognition results to generate parallel codes can improve the performance of parallel programs on heterogeneous multi-core processors significantly.

Key words: auto-parallelization; parallel cost model; heterogeneous multi-core; data transfer cost; accelerated parallel loop

0 引言

自动并行化是获得并行程序的一种简单高效的手段^[1]。无论是面向共享存储结构还是分布存储结构的自动并行化, 并行的主要对象都是程序中的循环。然而并不是所有循环都适合并行。循环并行化能否带来收益, 是判断一个循环是否进行并行的首要标准。当前大部分编译器和运行时库都包含一套收益评估机制, 即并行代价模型。它是自动并行化的核心, 是识别加速并行循环的基础和依据。准确高效的代价模型有利于提升所得并行程序的性能。

理想的并行代价模型应充分考虑计算机的硬件特征、程序特点及其相互影响, 并在尽可能短的时间内对并行代码做出尽可能精确的性能评估^[2]。为达到这个目标, 国内外众多学者展开了积极的探索与研究。针对共享存储系统, Trifunovic 等^[3]基于多面体模型构造了基于迭代空间和数组访问模式的代价函数来挖掘程序的细粒度并行性。Bondhugula 等^[4]提出基于循环结构分析的代价模型, 以获得最佳的循环合并方案, 增加程序并行粒度。OpenUH 编译器则把更多的影响因素考虑在内, 如同步开销、调度策略、负载均衡等, 通过建立更细致的模型为 OpenMP 程序提供更可靠的并行代价评估^[2]。分布存储系统方面, Sharapov 等^[5]结合排队理论及处理器和互连网络仿真技术, 预测 MPI/OpenMP

混合并行代码的性能, 挖掘系统的多级并行。这些模型都在一定程度上反映了系统的硬件特性和软件特点, 展现出一定的代价评估能力。但由于它们是针对特定体系结构设计的, 不适合直接在其他平台上使用。

异构多核处理器的出现使这个问题更加突出。它独特的处理核心结构、存储层次结构和通信机制使过去面向共享存储资源和对称计算资源的代价计算方法不再适用。新的体系结构要求编译器采用新的并行代价模型来衡量循环的并行收益, 从而给出正确的并行方案, 保证生成的并行程序具有较高的性能。

在此背景下, 本文设计并实现了面向异构多核处理器的并行代价模型。该模型以 Cell 处理器为目标结构, 定量地刻画了主从核不同运算能力、主局存不同访问延迟以及数据传输对循环并行执行效果的影响。通过比较循环的并行执行时间和串行执行时间, 判断循环并行化能否带来收益, 选择有收益的循环进行并行。实验结果表明, 该模型有效消除了无收益的并行循环, 显著改善了并行循环的识别效果, 提高了并行程序的性能。

1 异构多核结构

异构多核处理器是一种由结构不同、功能也不不同的处理器核心以某种方式互联而成的处理器组, 一般包含通用处理

收稿日期: 2012-12-13; 修回日期: 2013-02-20。 基金项目: 国家“核高基”重大专项(2009ZX01036-001-001-2)。

作者简介: 黄品丰(1987-), 男, 广东信宜人, 硕士研究生, 主要研究方向: 并行编译; 赵荣彩(1957-), 男, 河南洛阳人, 教授, 博士生导师, CCF 会员, 主要研究方向: 高性能计算、并行编译; 姚远(1972-), 男, 河南郑州人, 教授, 博士, 主要研究方向: 先进编译; 赵捷(1987-), 男, 内蒙古通辽人, 博士研究生, 主要研究方向: 先进编译。

核心和加速计算核心^[6]。典型代表有索尼、东芝和IBM公司联合开发的Cell处理器^[7-8]、Standard大学的Merrimac超级流计算机系统^[9]、ClearSpeed的CSX600处理器^[10]。其中Cell处理器的结构如图1所示。

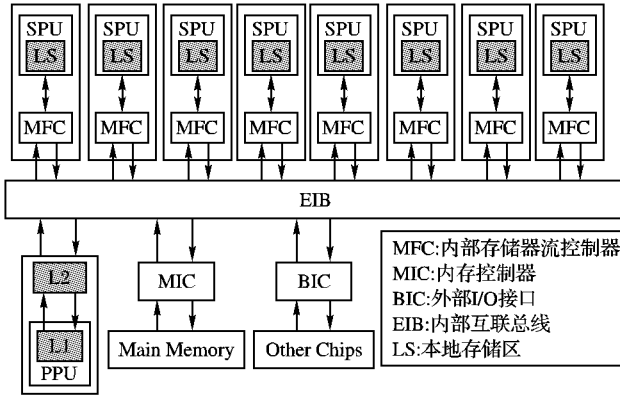


图1 Cell处理器结构

它由一个支持双线程的64位PowerPC通用处理器核PPE(Power Processor Element)和8个SIMD向量协处理器核SPE(Synergistic Processor Element)组成。PPE负责计算资源的管理和调度,SPE负责加速计算任务。PPE包含两级cache,SPE没有cache,却各有256KB的本地存储。SPE不能直接访问主存,必须通过直接存储访问(Direct Memory Access,DMA)操作完成系统主存与本地存储的数据传输。这些特点使Cell处理器在并行计算方面既有优势,又有劣势。优势是丰富的计算核心有利于获得更高的并行性能,劣势是当循环缺乏足够的并行粒度时,增加的数据传输开销和并行控制开销会使并行执行时间大于串行执行时间。因此,需要一个准确高效的并行代价模型,在并行化循环前评估循环的并行效果,从而做出正确的并行抉择,优化生成的并行程序性能。

2 面向异构多核结构的并行代价模型

并行化编译的首要任务是最大限度地挖掘串行程序的并行性。作为并行化的主要对象,循环嵌套常常通过循环交换、循环分布、循环合并等循环变换方法由不可并行循环变为可并行循环。然而并行是否有利,还需进一步的评估。

Open64编译器设计了一个并行代价模型,用于评估循环并行收益,但由于其面向共享存储结构,不能直接应用于异构多核平台。Blagojevic等^[11]提出MMGP(Model of Multi-Grain Parallelism)模型,用于预测并行程序在Cell处理器上的运行时间。该模型具有较高的准确度,但获得重要参数值的预运行(profiling)过程代价过高,不宜重复使用。

为了弥补以上不足,本文试图建立更加完善的并行代价模型,描述体系结构的变化对循环并行执行性能的影响。

2.1 循环的串行执行时间

循环串行执行时间是衡量循环并行收益的基准。在异构多核处理器中,循环串行执行时间是指循环嵌套在通用处理核心上的运行时间,主要包含运算指令执行时间、存储访问时间和循环控制开销。形式化地表示为:

$$T_{seq} = T_{opr} + T_{mem} + T_{loop} \quad (1)$$

1) T_{opr} 为循环体运算指令执行时间,它忽略了cache和旁路转换缓冲(Translation Lookaside Buffer, TLB)失效开销,仅考虑条件就绪时指令的运算部件执行周期。因此,它与指令种类、操作数类型以及处理器运算能力密切相关。在编译过程中,编译器可通过分析循环的中间语言表示获得指令及其

操作数的相关信息,根据这些信息和Cell处理器的计算能力,得到相应指令的执行周期。 T_{opr} 即为循环体一次迭代的指令运算时间乘以循环迭代次数。

$$T_{opr} = T_{machine_per_iter} * N_{loop_iter} \quad (2)$$

2) T_{mem} 为存储访问时间,等于平均访存时间乘以访存次数。在Cell处理器中,由于PPE具有两级cache,平均访存时间等于cache命中时间加上cache失效率乘以失效开销。当二级cache失效时,失效开销即主存访问延迟。如式(3)所示, T_{L1} 、 T_{L2} 为两级cache命中时间, F_{L1} 、 F_{L2} 为两级cache失效率, T_{main_mem} 为主存访问时间, N_{mem} 为访存次数。编译时,编译器可遍历循环的中间语言表示,分析循环的所有访存要求和访问模式,结合PPE的cache大小和cache配置,估计cache的失效率。如访问 $a[i][j]$ 、 $a[i][j+1]$ 、 $a[i][j+2]$ 等同一数组的相邻元素时只计算一次失效。

$$T_{mem} = (T_{L1} + F_{L1} * (T_{L2} + F_{L2} * T_{main_mem})) * N_{mem} \quad (3)$$

3) T_{loop} 为循环控制开销,主要指循环变量赋初值、循环条件判断、循环变量增值、循环入口跳回等指令的执行时间。精确地估计这个时间是比较困难的,因为循环展开、循环分段等循环优化会改变初始的循环控制结构,而且跳转指令可能导致的cache失效也是不可预计的。因此令每次循环迭代的控制开销为一常数值,该值可通过在Cell处理器上运行简单的循环测定。

$$T_{loop} = T_{overhead_per_iter} * N_{loop_iter} \quad (4)$$

2.2 循环的并行执行时间

循环在异构多核处理器上的并行执行方式可简单描述为:先由通用处理核心通过DMA操作把代码和数据从主存传输到加速计算核心的本地存储,再由加速计算核心进行并行计算,最后把计算结果返回主存或通用处理核心。如图2所示。

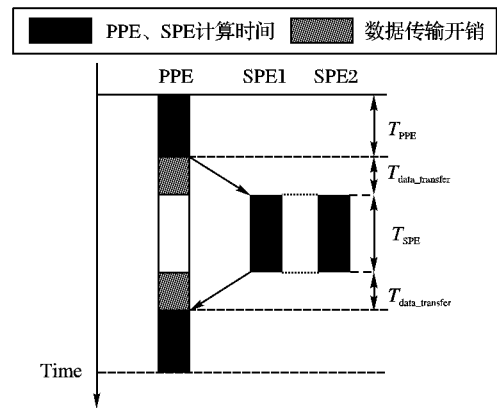


图2 循环并行执行方式

可见循环的并行执行时间主要由三部分组成:数据传输开销、并行控制开销和并行计算时间。形式化地表示为:

$$T_{para} = T_{data_transfer} + T_{para_overhead} + T_{compute} \quad (5)$$

2.2.1 数据传输开销

在Cell处理器中,SPE不能直接访问主存,需要通过DMA操作把计算所需的数据提前拷贝到本地存储,并在计算结束后把结果发送回主存。这过程消耗的时间即数据传输开销。实际应用中,常常把循环内向上暴露引用的变量和定义的变量在主存和局存之间来回传输。在总线带宽一定的情况下,传输的数据量决定了传输的开销。如式(6)所示:

$$T_{data_transfer} = C_s + C_t * \left(\sum_{x \in use(l)} x.size + \sum_{y \in def(l)} y.size \right) \quad (6)$$

其中: C_s 为 DMA 操作的初始化时间, C_t 为每字节数据的传输时间, $use(l)$ 为循环 l 中向上暴露引用的变量集合, $def(l)$ 为循环 l 内定义的变量集合, $x.size$ 、 $y.size$ 为变量所占存储空间。编译时, 编译器可通过数据流分析求得循环内定义和引用的变量集合, 并通过变量类型得知变量所占字节。

2.2.2 并行控制开销

并行计算需要必要的启动和管理开销, 统称并行控制开销。可以简单地用以下公式计算:

$$T_{para_overhead} = T_c + p * T_p \quad (7)$$

其中: T_c 为并行启动开销, T_p 为单个计算核心的并行管理开销, p 为使用的计算核心数。 T_c 、 T_p 可通过运行简单的循环测定, 根据测量结果赋予合适的常数值。

2.2.3 并行计算开销

并行计算开销等于循环在单核上的运行时间除以计算所用核心数。单核运行时间的计算过程与主核类似, 但由于从核的结构和性能与主核不同, 各部分消耗的时间也不同。如式(8)所示:

$$T_{compute} = (T'_{opr} + T'_{mem} + T'_{loop}) / p \quad (8)$$

其中: T'_{opr} 、 T'_{mem} 、 T'_{loop} 分别为从核的运算指令执行时间、存储访问时间、循环控制开销。因为从核没有 cache, 只有软件管理的本地存储, 所以 T'_{mem} 的计算方式有所不同。如式(9)所示:

$$T'_{mem} = T_{ls} * N_{ls} \quad (9)$$

其中: T_{ls} 为本地存储访问延迟, N_{ls} 为本地存储访问次数。

2.3 并行收益评估

当循环的并行执行时间小于串行执行时间时, 并行化获得收益, 标记此循环为并行循环。

$$T_{data_transfer} + T_{para_overhead} + T_{compute} < T_{opr} + T_{mem} + T_{loop} \quad (10)$$

式(10)为并行收益的判断标准。它之前推导的公式构成完整的并行代价模型。该模型较好地反映了异构多核处理器的特点, 具体地刻画了主核从核不同计算能力、不同存储结构对循环并行执行方式和执行性能的影响。可依据此模型识别串行程序中的加速并行循环, 提高生成的并行代码性能。

3 加速并行循环识别算法

并行代价模型的作用是在并行循环前, 评估循环的并行收益, 为并行决策提供有益的参考。出于此目的, 我们提出了基于并行代价模型的加速并行循环识别算法。该算法如下所示。

算法 基于并行代价模型的加速并行循环识别算法。

输入 串行程序 F ;

输出 能够实现加速的并行循环

```

1) procedure Parallel_Loop_Recognition
2)    $F' = \text{Loop\_Optimization}(F)$ 
3)   for every loop  $l$  in program  $F'$ 
4)     if ( $l$  is Parallelizable) then
5)       if ( $\text{parallel\_cost}(l) < \text{Sequential\_Cost}(l)$ ) then
6)         mark  $l$  as loop to be executed concurrently
7)       else mark  $l$  as loop to be executed sequentially
8)     else continued
9)   end for
10) end procedure
11) subroutine sequential_cost( $l$ )
12)   obtain  $N_{loop\_iter}, F_{L1}, F_{L2}, N_{mem}$  through code analysis
13)    $\text{sequential\_cost} = T_{machine\_per\_iter} * N_{loop\_iter} + (T_{L1} + F_{L1} * (T_{L2} + F_{L2} * T_{main\_mem})) * N_{mem} + T_{overhead\_per\_iter} * N_{loop\_iter}$ 

```

```

 $N_{loop\_iter}$ 
14) return sequential_cost
15) end subroutine
16) subroutine parallel_cost( $l$ )
17)   obtain use( $l$ ) and def( $l$ ) through data flow analysis
18)   estimate  $N_{ls}, N_{loop\_iter}$  through code analysis
19)    $\text{parallel\_cost} = C_s + C_t * (\text{sum of } x.size + \text{sum of } y.size) + T_c + p * T_p + (T'_{machine\_per\_iter} * N_{loop\_iter} + T_{ls} * N_{ls} + T'_{overhead\_per\_iter} * N_{loop\_iter}) / p$ 
        $x$  belongs to use( $l$ ) and  $y$  belongs to def( $l$ )
20) return parallel_cost
21) end subroutine

```

首先对串行程序进行循环优化, 挖掘程序中潜在的并行性(第2行), 然后遍历程序中的每个循环, 如果循环可并行, 则分别计算循环的并行执行时间和串行执行时间, 若并行执行时间小于串行执行时间, 则认为此循环并行化有收益, 标记此循环为并行循环; 否则令其串行执行(第3至9行)。在计算并行执行时间和串行执行时间的过程中, 可通过查看处理器规范获得平台相关参数, 并通过中间语言表示分析获得循环的基本信息。

本算法已在开源编译系统 Open64 中实现。在 Open64 的 LNO 阶段, 通过标量扩展、循环交换、循环分布等循环优化方法尽可能地把程序中的不可并行循环转换为可并行循环。在 Perform_ARA_and_Parallelization 函数, 遍历优化后的程序循环, 对于可并行的循环, 使用并行代价模型判断并行是否有利, 从而决定是否插入 OpenMP 并行编译指示。在 whirl2c/whirl2f 阶段根据算法的识别结果生成 OpenMP 并行代码。由于编译器在并行化循环之前有一个评估过程, 力图避免负收益循环的并行, 使生成的并行程序获得高性能成为可能。

4 实验评估

4.1 实验平台

为验证本文模型对异构多核结构上加速并行循环的识别效果, 我们在 IBM QS20 刀片服务器上进行测试。其一块刀片包含两个主频 3.2 GHz 的 Cell 处理器, 主存 1 GB。操作系统是 Fedora core 9.0, 安装基于 Open64 5.0 的含有本文并行代价模型及优化算法的自动并行工具, 基础编译器为 IBM XL 单源编译器。

4.2 测试用例

测试采用 NPB3.3.1 基准测试集中的 8 个程序 BT、CG、EP、FT、IS、LU、MG 和 SP。NPB(NAS Parallel Benchmark)是由美国国家航空航天局开发的一个小型程序集^[12], 其中的程序来自流体动力学应用领域, 可以表现出一般应用程序的实际性能, 目前已普遍作为高性能计算的性能测试标准。按数据规模的大小由小到大可分为 S、W、A、B、C 五个级别。

4.3 测试方法

首先使用自动并行工具识别串行程序中的加速并行循环, 计算所用时间占总编译时间的百分比, 以之评估并行代价模型的效率。然后, 对比自动并行工具的识别结果和实测结果, 评估并行代价模型的正确性。最后, 使用基础编译器编译生成的并行程序, 比较优化前后生成的并行程序性能。

4.4 实验结果及分析

表1显示了自动并行化过程中, 使用并行代价模型识别串行程序加速并行循环所用时间占总编译时间的百分比。所测的 8 个程序, 平均百分比为 1.25%。由此可见, 并行代价模型的调用不会大幅增加编译负担。

表1 各程序调用模型时间占总编译时间百分比 %

测试程序	调用代价模型时间占总编译时间百分比	测试程序	调用代价模型时间占总编译时间百分比
BT	1.17	IS	1.27
CG	1.25	LU	1.31
EP	1.32	MG	1.20
FT	1.23	SP	1.26

本文以 EP 程序为例说明并行代价模型对加速并行循环的识别效果。在 EP 的 8 个循环嵌套中,循环 2,4,5,7,8 由于计算量过小,虽然可并行,但实际上无法通过并行获得收益。因此并行代价模型将其识别为串行。与 Open64 原有代价模型相比,本文模型减少了负收益循环的并行,更适用于异构多核平台。如表 2 所示。

表2 各方式对 EP 程序加速并行循环的识别情况

循环编号	实测结果	Open64 模型	本文模型
1	串行	串行	串行
2	串行	并行	串行
3	串行	串行	串行
4	串行	并行	串行
5	串行	并行	串行
6	并行	并行	并行
7	串行	并行	串行
8	串行	并行	串行

对 NPB3.3.1 基准测试集其他 7 个程序的测试也获得了相似的测试结果,如表 3 所示。本文模型识别出的可实现加速的并行循环个数基本达到实测水平,与 Open64 的识别结果相比平均提高约 38%。

表3 各方式对程序加速并行循环的识别效果

程序	循环个数	Open64 并行数	本文模型 并行数	实测 并行数	改进的 百分比/%
BT	51	17	12	10	29.41
CG	25	11	6	5	45.45
EP	8	6	1	1	83.33
FT	26	4	3	3	25.00
IS	13	3	2	2	33.33
LU	55	27	21	18	22.22
MG	30	12	7	5	41.67
SP	56	21	16	13	23.81

本文以并行代价模型识别结果为依据,生成异构多核并行代码,并测试其在 Cell 处理器上的性能。实验结果表明,基于本文并行代价模型的并行代码性能要优于基于传统并行代价模型的并行代码性能。如图 3 所示,在 C 规模、8 线程时,8 个程序的平均性能提升达到 22.1%。

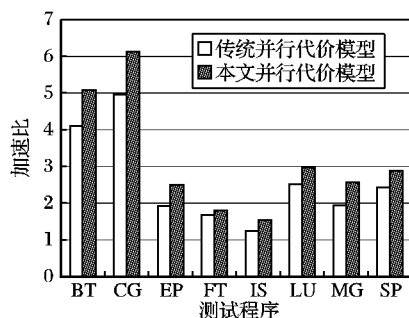


图3 两种模型生成的并行程序性能对比

与传统并行代价模型相比,本文并行代价模型更多地考

虑了异构多核架构的硬件细节,引入的数据传输开销、并行控制开销和并行计算开销,有效地刻画了异构结构对循环并行执行时间的影响,大大提高了并行循环识别的准确性,减少了负收益循环的并行,从而提升生成的并行代码性能。

5 结语

本文面向异构多核处理器建立了一个新的并行代价模型,并基于该模型设计了一个加速并行循环识别算法,以提高自动并行化时并行循环识别的准确性。实验结果表明,本文提出的并行代价模型是准确高效的,把它的并行识别结果作为后端并行代码生成的依据能有效提高并行程序的性能。下一步工作是分析如何估算并行计算与数据传输重叠时循环的并行执行时间,以进一步提高并行代价模型的识别能力。

参考文献:

- [1] 沈志宇, 胡子昂, 廖湘科, 等. 并行编译方法[M]. 北京: 国防工业出版社, 2000.
- [2] LIAO C H. A compile-time OpenMP cost model[D]. Houston: University of Houston, 2007.
- [3] TRIFUNOVIC K, NUZMAN D, COHEN A, *et al.* Polyhedral-model guided loop-nest auto-vectorization[C]// Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques. Washington, DC: IEEE Computer Society, 2009: 327-337.
- [4] BONDHUGULA U, GUNLUK O, DASH S, *et al.* A model for fusion and code motion in an automatic parallelizing compiler[C]// Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques. Washington, DC: IEEE Computer Society, 2010: 343-352.
- [5] SHARAPOV I, KROEGER R, DELAMATER G, *et al.* A case study in top-down performance estimation for a large-scale parallel application[C]// Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM, 2006: 81-89.
- [6] CONG J, YUAN B. Energy-efficient scheduling on heterogeneous multi-core architecture[C]// Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design. New York: ACM, 2012: 345-350.
- [7] CHEN T, RAGHAVAN R, DALE J N, *et al.* Cell broadband engine architecture and its first implementation - a performance view[J]. IBM Journal of Research and Development, 2007, 51(5): 559-572.
- [8] SKOVHEDE K, LARSEN M N, VINTER B. Extending distributed shared memory for the cell broadband engine to a channel model[C]// Proceedings of the 10th International Conference on Applied Parallel and Scientific Computing. Berlin: Springer-Verlag, 2012, 7133: 108-118.
- [9] UJVAL J K, RIXNER S, WILLIAN J D, *et al.* Programmable stream processors[J]. Computer, 2003, 36(8): 54-62.
- [10] KINDRATENKO V V. Novel computing architecture[J]. Computing in Science & Engineering, 2009, 11(3): 54-57.
- [11] BLAGOJEVIC F, FENG X Z, CAMERON K W, *et al.* Modeling multigrain parallelism on heterogeneous multi-core processors: a case study of the cell BE[C]// Proceedings of the 2008 International Conference on High-Performance Embedded Architectures and Computers. Berlin: Springer, 2008: 38-52.
- [12] SHAN H Z, BLAGOJEVIC F, MI S J, *et al.* A programming model performance study using the NAS parallel benchmarks[J]. Scientific Programming, 2010, 18(3/4): 153-167.