面向多核处理器的自动并行化技术展望

赵银亮
西安交通大学
2011年4月20日

面向多核处理器的自动并行化技术展望

"自动并行化"

"自动控制"

并行：
parallel（并行的）
parallelism（并行性、并行度）
parallelization（并行化）
parallelize（使并行）

研究动机

- 开发和利用并行性能够提高计算系统的运行速度，从而缩短问题求解时间。
  - 计算机系统领域，并行处理是技术基础之一
  - 计算科学中，并行计算是决定问题求解时间长短、甚至能否求解的决定性因素。
- 片上多处理器（**CMP**）出现后产生了新的现象
  - 计算机用户不得不去关心并行性，甚至与并行打交道。

自动与手工并行化

- 手工并行化
  - 人针对并行机开发并行应用程序。
  - 不是人人都能够做好的事情。
- 自动并行化
  - 计算机软硬件开发和利用应用程序中的并行性
  - 使得并行性开发和利用这件事能为人人所受益、为事事所受益。
  - 实现了**CMP**的性能指标的确定化，保持摩尔定律继续成立。

多核处理器为自动并行化带来机遇

- 一种普遍观点：
  - 自动并行化效率低，难以实用
  - 只局限在循环级并行化以及科学计算领域
- 新的希望：面向多核处理器的线程级并行化
  - 这是本讲座想要得到的结论

本讲座内容提纲

- 开发和利用并行性的普遍化现象
- 自动并行化与手工并行化孰优孰劣
- 自动并行化，软件做多少？硬件做多少？
- Prophet编译系统：一个自动并行化的实例
- 技术展望及总结

一、并行普遍化现象分析

- 概念和术语
- 开发和利用并行性的意义
- 人人参与并行的原因

1.1 概念和术语

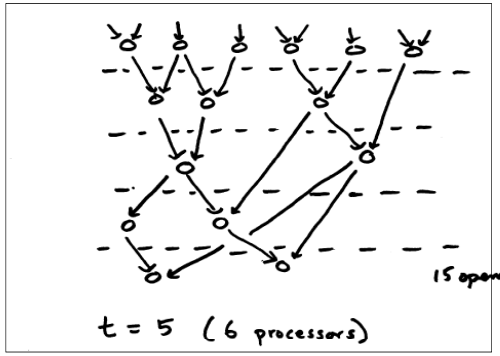
- 并行性是指属于问题求解过程的一些操作同时执行

a:=b+1
x:=y+e
w:=a+x

⇨

a:=b+1 x:=y+e
w:=a+x

T_1=3; T_p=2; G=1.5; S=1.5



To schedule on 2 processors,

1st level	time 3
2nd level	time 2
3rd level	time 1
4th level	time 1
5th level	time 1
	8

Brent

$t_2 \le 5 + \frac{15-5}{2}$

$t_2 \le 10$

So Brent's bound is not tight

循环级并行示例

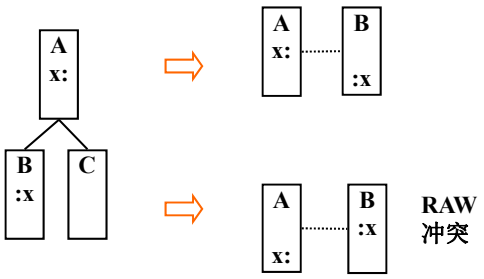
```
do i=1, n
   z(i) = x(i) + y(i)
enddo
```

```
do i=2, n
   z(i) = z(i-1)*2
enddo
```

⇨

```
do i=2, n
   z(i) = z(1)*2**(i-1)
enddo
```

线程级并行示例



T_1=2; T_p=1; G=2; S=2

并行性等级

- 指令内部
- 指令之间
- 线程级
- 进程级

应用程序的并行度

- **Amdahl定理**

$$S = \frac{1}{f + \frac{1-f}{p}}$$

f 问题中的串行分量

p 处理单元个数

从Amdahl定理看并行应用

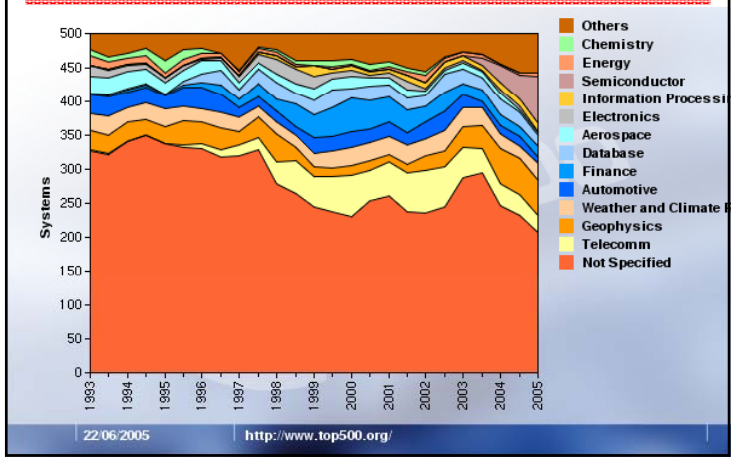
- **Embarrassingly parallel problems**
  - 并发单位之间没有依赖
  - $f \to 0$
  - 图形学、人脸识别、遗传算法、**BLAST**
- 规则并行问题
  - 大部分科学计算问题
  - f 小
- 非规则并行问题
  - 数据与通信结构相关
  - f 相当大
  - 排序、稀疏矩阵、链表、树、图

1.2 开发和利用并行性的意义

- 对于问题求解而言
  - 缩短应用程序的执行时间
  - 决定可解问题规模
  - 决定问题是否可解

- 在计算科学中，永远存在大量应用，因为时间因素不能解决。
- 人们期待它们得到求解的热情永远不会减退。
- 求解问题所花费的时间总是可以缩短的。

高端并行机的应用领域



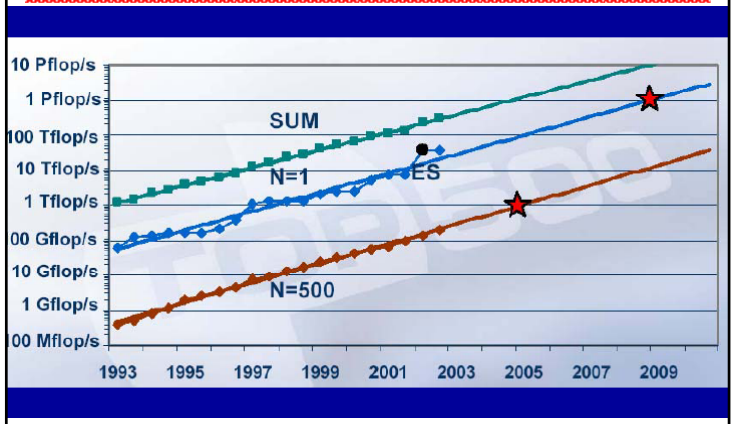
高端并行机的性能

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz , Voltaire Infiniband / 2008 IBM	129600	1105.00	1456.70	2483.47
2	Oak Ridge National Laboratory United States	Jaguar - Cray XT5 QC 2.3 GHz / 2008 Cray Inc.	150152	1059.00	1381.40	6950.60
3	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX, Xeon QC 3.0/2.66 GHz / 2008 SGI	51200	487.01	608.83	2090.00
4	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution / 2007 IBM	212992	478.20	596.38	2329.60
5	Argonne National Laboratory United States	Blue Gene/P Solution / 2007 IBM	163840	450.30	557.06	1260.00
6	Texas Advanced Computing Center/Univ. of Texas United States	Ranger - SunBlade x6420, Opteron QC 2.3 Ghz, Infiniband / 2008 Sun Microsystems	62976	433.20	579.38	2000.00
7	NERSC/LBNL United States	Franklin - Cray XT4 QuadCore 2.3 GHz / 2008 Cray Inc.	38642	266.30	355.51	1150.00
8	Oak Ridge National Laboratory United States	Jaguar - Cray XT4 QuadCore 2.1 GHz / 2008 Cray Inc.	30976	205.00	260.20	1580.71
9	NNSA/Sandia National Laboratories United States	Red Storm - Sandia/ Cray Red Storm, XT3/4, 2.4/2.2 GHz dual/quad core / 2008 Cray Inc.	38208	204.20	284.00	2506.00

>1 petaflop!

> 100K cores

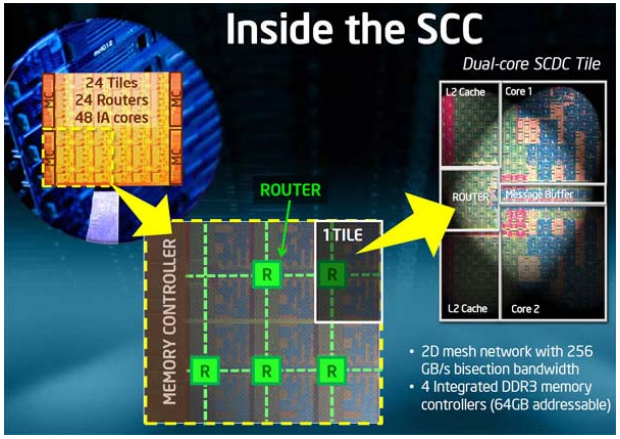
性能指标趋势预测



开发和利用并行性的意义（续）

- 对于计算机系统而言
  - 决定系统性能（计算速度）
  - 决定系统响应时间
- 对于片上多处理器系统而言
  - 多核处理器单DIE上封装多个CPU核，一般从2个到几十个。如AMD Opteron有2-4个核。
  - 众核处理器是指在多个单DIE多核或单DIE单核之间形成互联结构，这使处理器中核的数量增加到成百上千个。如Intel SCC有48个核。

图：多核



1.3普及并行的原因分析

- 使用16核的多核处理器，运行某个串行应用程序，这个程序的执行时间T_{16}应该比串行执行时间T_1短，为什么只能达到T_1而不是T_{16}？
- 如果运行并行应用程序，那么能否正确运行？若能，加速比S如何？

- 尽管开发和利用并行性很有意义，但人人参与并不是好事，从技术发展规律来看，一段时间之后，需要把一批人解放出来。

二、自动并行化与手工并行化孰优孰劣

- 手工并行化
  - 优点与挑战
- 自动并行化
  - 优点与挑战
- 二者对比怎样？

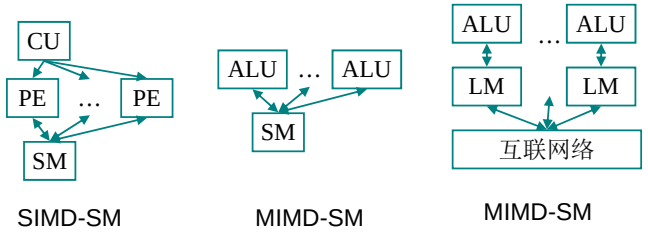
2.1 手工并行化？

- 手工并行化是指对于给定问题，针对某并行机写出并行算法并采用某并行语言编程实现。
- 优点
  - 效率高。
- 挑战
  - 如何恰当地将并行机、并行算法和并行编程匹配起来，取得解决这个问题的理想的效果。

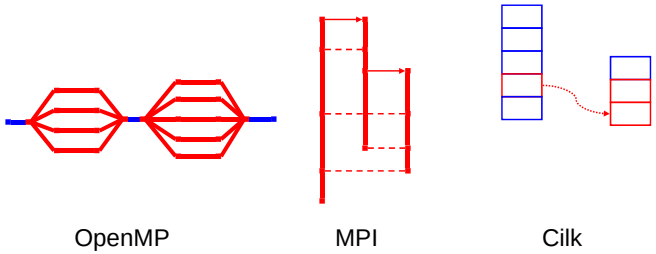
一般应对方法

- 并行机多样化、并行语言机制的多样化，并行算法设计复杂性以及并行编程费时费力的特点等共同造成手工并行化非易事。
- 解决途径，通过层层抽象，在一定抽象层次上，将并行机、并行编程、并行算法统一起来，这有助于开发出理想的并行应用程序。

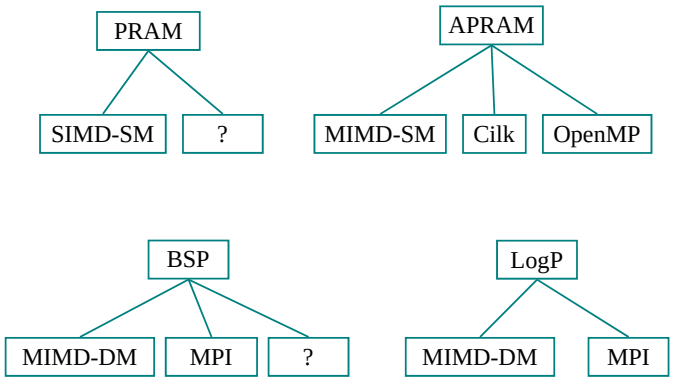
并行机模型示意



并行编程模型示意



并行计算模型示意



观察

- 开发并行应用程序需要考虑对多样化因素的适应性。
- 目前还没有系统的开发方法来提高开发效率。
- 在规则问题的求解方面，手工并行化仍然是主流技术。

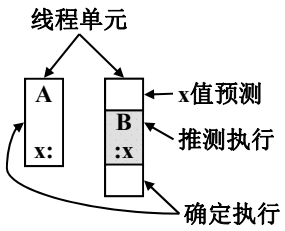
2.2 自动并行化？

- 对给定问题的串行程序，由编译器转换成并行目标代码，在硬件支持的目标机上执行。
- 自动并行化的好处是明显的，避免了人去开发并行应用程序。

续

- 目前在有限范围或模型上可做，如CPU核内部、MIMD-SM或APRAM模型中等。
  - ILP级别上的隐式并行技术已经比较成熟，如流水线、超标量、乱序执行以及VLIW等。
  - 基于Fortran的对循环并行化（数据并行）有效，当然并行化循环仍然是难点，所以主要在Fortran中取得了理想效果。
  - 线程级并行性（TLP）的开发和利用产生出推测多线程（SpMT），编译器完成线程划分，硬件支持线程级推测执行。

推测多线程示例



自动并行化的难点

- 难点：程序分析
  - 如何准确识别出并发单位 （程序的动态性不可预测性造成静态分析存在较大误差，程序输入域不可知造成控制流预测的结果不确定。 ）
  - 并发单位是否安全 （涉及到数据依赖分析和并发单位间数据依赖的动态消解。 ）
  - 是否值得并行化 （期望加速比不理想、并行开销在执行时间中占比过大 ）

2.3 对比

等级	自动并行化	手工并行化	对比结果
指令级	流水线、超标量、乱序执行； VLIW	汇编级编程	完全自动并行，很有效果
循环级	自动识别循环及循环优化	HPF	相结合方式，有效
线程级	SpMT	OpenMP、Pthread、Cilk	一般不结合，自动并行略逊
进程级	并发性分析；Paradigm编译器	MPI	自动并行差距很大；

观察

- 自动并行化在指令级和循环级取得了与显式并行相媲美的效果，所以几乎没有手工并行化；
- 多核处理器为线程级自动并行化提供了平台，其自动并行化效果越来越明显；
- 在进程级，自动并行化目前没有有效方法，这也限制了自动并行只在共享存储器并行机模型可用。

三、自动并行化的软硬件分工

- 自动并行化一般由软、硬件配合来完成，其中编译器（或专门工具软件）负责从串行程序中识别并发单位，这需要建立在对程序进行分析的基础上，而硬件支持实现自动并行化也是必不可少的。

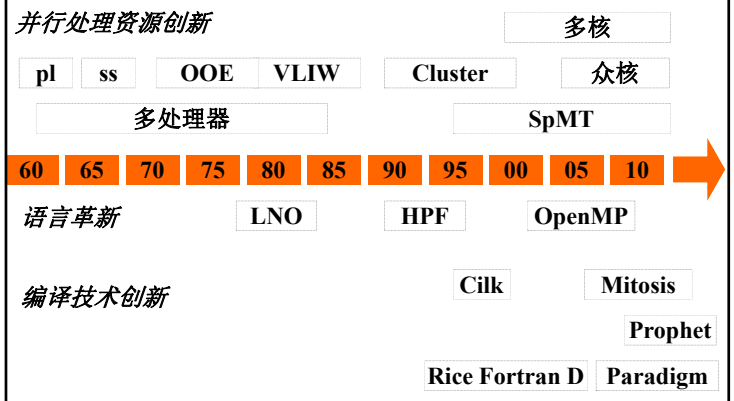
3.1 自动并行化发展历程概述

- ILP中，如流水线、超标量、乱序执行等，主要由硬件做指令间依赖分析和指令调度，并在硬件支持下提高指令执行的吞吐率；编译器所做的依赖分析和指令调度仅起到进一步优化作用。
- 对于VLIW，依赖分析和指令调度只由编译器做而硬件做VLIW指令拼装。其目的是在多CPU上达到并行执行指令的效果。
  - Lisp机Lisp_M1系统

- 循环级并行中，编译器从串行程序中识别循环依赖关系；目标机为多处理器系统。
  - 对应于PRAM和APRAM模型。
  - 不会被看作是提高处理器性能的手段。
- 程序分析的非精确性限制了循环级自动并行化的有效性
  - HPF；Rice Fortran D
  - 与手工并行结合，如OpenMP

- 多核处理器中，CPU核间的线路距离更短，其寄存器堆间的数据迁移时间更短、保持Cache一致性的时间更短，因而从硬件支持上来看大大减小了线程发起、撤销、线程间同步、通信的开销，使得推测多线程（SpMT）技术应运而生。
- SpMT软硬件首次实现了完整的线程级自动并行化。

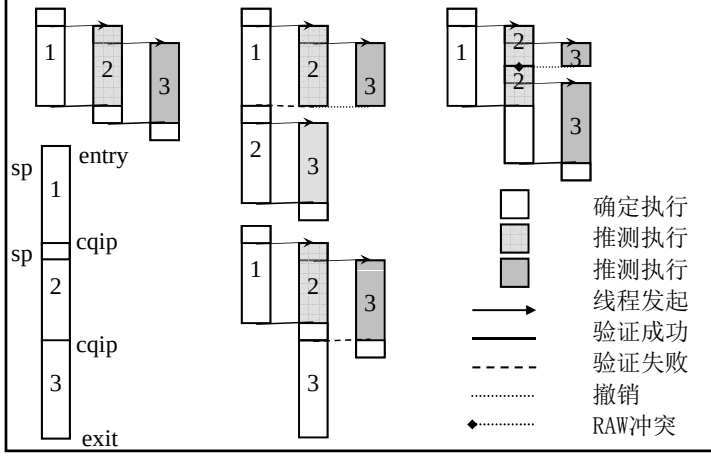
自动并行化发展历程



3.2 SpMT技术与多核处理器的加速比性能

- CPU性能评测
  - 串行基准程序
- 多核处理器速度是多少？
  - 串行程序无法利用多核的并行处理资源
  - 逐一先并行化再运行是不可能的
  - 自动并行化是必然选择
  - 目前仅有SpMT技术合适

SpMT原理示意



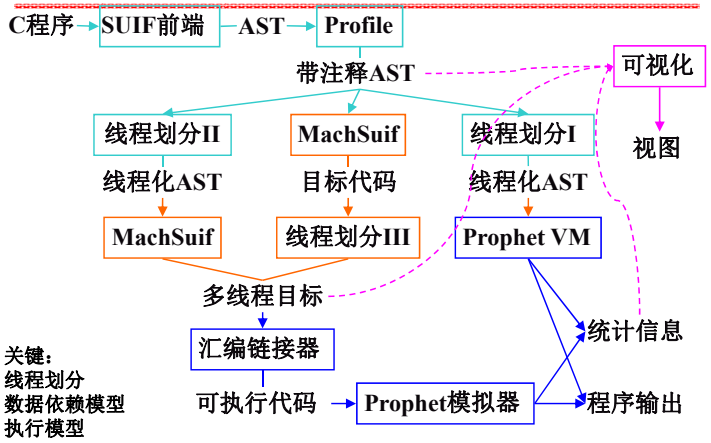
四、研究实例：Prophet编译系统

- 项目背景
  - 863课题：面向CMP的推测多线程编译技术研究
- 课题目标
  - 面向多核处理器结构自动并行化C程序，产生SpMT多线程目标代码，并通过模拟器运行和性能评测。

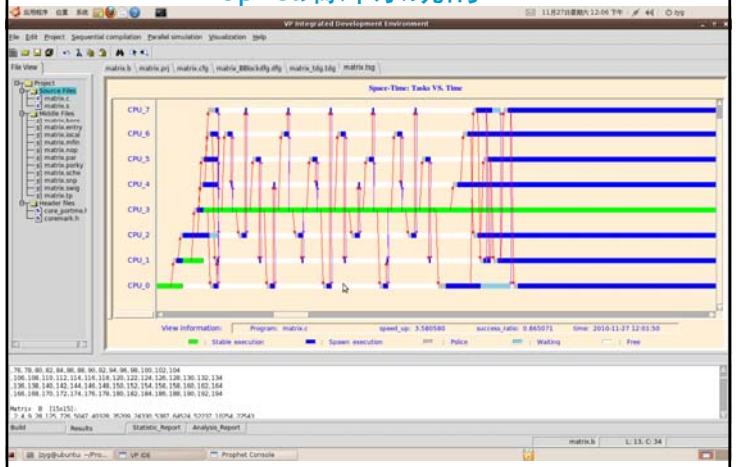
系统概述

- Prophet编译系统由编译器和模拟器组成。
  - Prophet编译器是在SUIF编译框架上增加了将C程序转换为SpMT多线程目标代码的功能。
  - Prophet模拟器扩展MIPS指令集实现了SpMT执行模型，以模拟硬件支持SpMT。
- Prophet编译系统并行化效果
  - Olden基准程序测试，对非规则程序的加速比1.7以上
  - 规则程序加速比8核达到5.8

Prophet编译系统的编译过程



Prophet编译系统的IDE



Prophet执行模型的特点

- 采用预计算技术支持线程代码段的活跃变量集的值预测，为纯软件方式，实现方便但与硬件实现方式相比值预测效率较逊色。
- 采用多版本Cache支持推测执行、冲突检测和结果提交。
- 采用硬件动态检测RAW冲突。
- 有单核单线程和单核多线程执行模式，便于分析CPU核利用率与加速比间的关系。
- 有共享栈和独立栈两种运行时栈管理模式，便于分析函数调用的并行化效果。

Prophet编译器的特点

- 在MIPS指令集中扩充了8条支持SpMT的指令。
- 多种线程划分方案可选，便于对比不同策略之间的优化效果。
- 采用基于纯软件Profile的分支预测机制，能够考虑到程序输入集在分支预测中的主要作用，但比硬件值预测效率方面略逊色。
- 代码移动机制降低了RAW冲突发生概率。
- 以线程代码段为单位的寄存器分配提高了寄存器使用效率

Olden基准程序

Princeton大学开发的测试集，主要采用链表和树等动态数据结构和递归控制结构。是相关研究领域公认的基准测试程序。

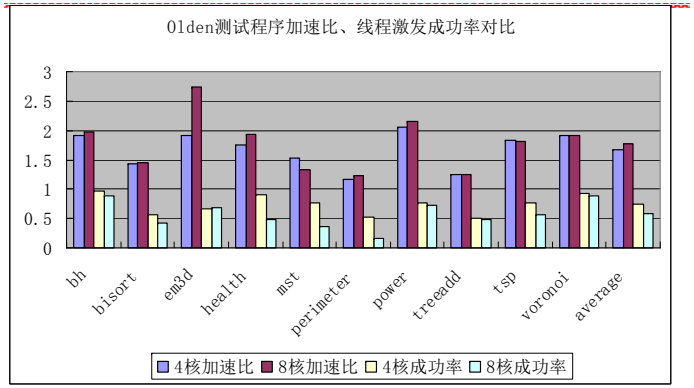
程序名	应用领域描述	主要数据结构
mst	计算图的最小生成树	单向链表
bh	巴恩斯-胡特N体模拟算法	异构八叉树
power	求解电力定价制度优化问题，叶子节点为客户要求权力和根节点为电厂返回定价	N叉树、单向链表
tsp	用分割算法和启发式最近点解决“货郎担”问题	平衡二叉树
em3d	模拟三位物体中的电磁波传播	单向链表
perimeter	计算图像区域的周长	四叉树
bisort	整数双调排序问题	二叉树
voronoi	用递归遍历树中结点集来解决voronoi图的问题	平衡二叉树
treeadd	将二叉树中所有节点求和	二叉树
health	模拟Columbian保健医疗系统	双向链表

Prophet模拟器配置参数

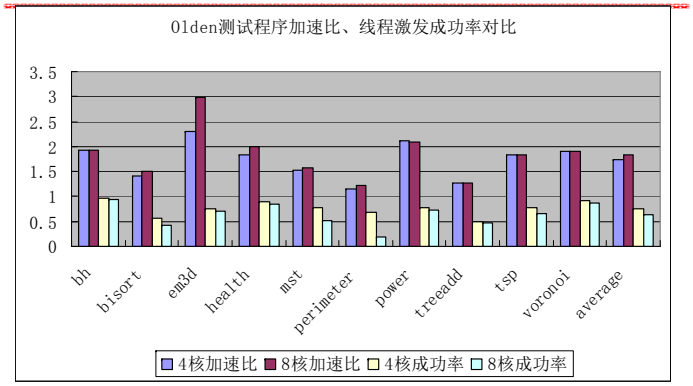
Prophet模拟器模拟的是超标量流水线多核处理器。包含ALU部件、Cache部件、流水线部件。具体的配置信息见下表：

配置项	项目值
发射/提交宽度	4
取指令队列	4
顺序发射	是
ALU功能单元（整数、浮点）	ALU(4个) 1个时钟周期、MULT(1个) 4个时钟周期、DIV(1个) 12个时钟周期
多版本 Cache	4路组相联、64KB，命中延迟：2个时钟周期
激发开销	5个时钟周期
验证开销	15个时钟周期
本地寄存器	1个时钟周期
访问内存	5个时钟周期
提交开销	5个时钟周期

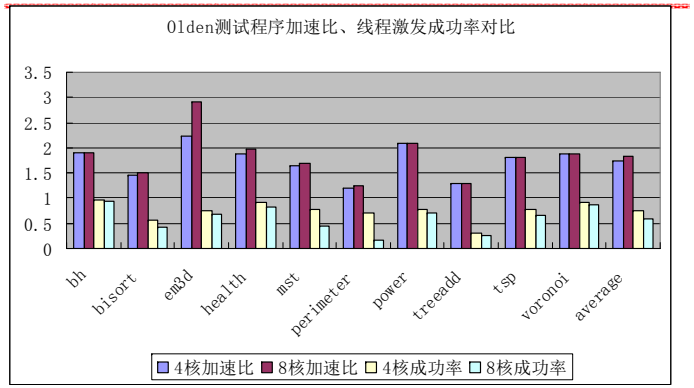
前端共享栈模式测试性能



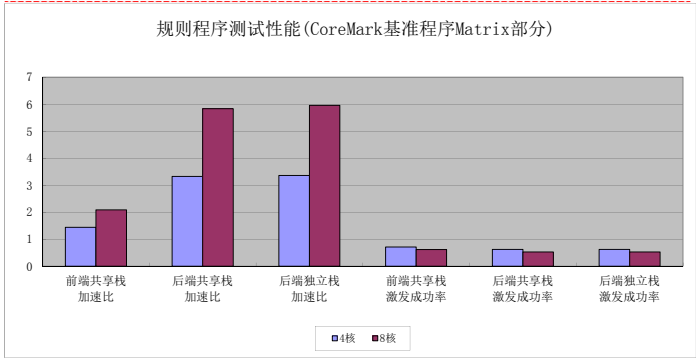
后端共享栈模式测试性能



后端独立栈模式测试性能



规则程序测试



观察

- 提供了可行的SpMT软硬件功能分配方案
- 对加速非规则程序和规则程序均有效
- 随着核数目由4核增加到8核，加速比略有增加，而推测执行成功率均降低
- 软硬件协同设计是解决SpMT并行化瓶颈的有效手段

专利申请

- 编译器支持下的推测多线程内存数据同步执行方法及装置，专利申请号201010160536.1，2010.4
- 基于CMP的推测多线程机制下的独立栈函数调用方法，专利申请号201010186162.0 2010.5
- 一种动态多线程划分方法，专利申请号201010230483.6，2010.7
- 一种面向粒的编程集成开发平台，专利申请号201010172797.5，2010.5
- 一种面向粒的程序构造方法，专利申请号201010172831.9，2010.5

软件著作权

- 基于CMP和推测多线程执行模型的Prophet模拟器软件，受理号2010R11S030075
- 基于CMP和推测多线程执行模型的Prophet编译系统，受理号2010R11L027863
- 基于CMP和推测多线程执行模型的Prophet虚拟机软件，受理号2010R11L027883
- 支持Java程序推测并行执行的Rope Java虚拟机软件，受理号2010R11L027707
- 基于推测多线程编译技术的可视化集成开发环境VPIDE软件，受理号 2010R11L073006

论文发表

An Overview of Prophet. ICA3PP 2009: 396-407

A Thread Partition Method for Speculative Multithreading. IEEE ScalCom-EmbeddedCom 2009 : 285-290

Prophet: A Speculative Multithreading Execution Model with Architectural Support Based on CMP. IEEE ScalCom-EmbeddedCom 2009 : 103-108

A Cost Estimation Model for Speculative Thread Partitioning. IEEE ISPA 2010, pp: 66 - 74

Optimistic Parallelism Based on Speculative Asynchronous Messages Passing. IEEE ISPA 2010,pp: 382 - 391

Prophet Synchronization Thread Model and Compiler Support. IEEE ISPA 2010, pp:81-87

A Cost Estimation Based Speculative Path Determination Method for Speculative Thread Partitioning. IEEE ICCASM 2010, Vol.3, pp:663-668

支持推测多线程的扩展多核模拟器Prophet．西安交通大学学报. 44(10): 13-17, 2010

一种基于代价评估的推测多线程路径预测方法．西安交通大学学报. 44(12):22-27, 2010

基于CMP的推测多线程机制下的独立栈模型．西安交通大学学报. 44(12) :10-15, 2010

Prophet推测多线程系统设计与实现. 计算机科学. 2011,38(2):296-301

一种基于推测多线程技术的程序可视化方法与实现．西安邮电学院学报. 15(5): 69-74，2010

线程划分中的寄存器依赖问题分析．微电子学与计算机（已录用）

五、结束语及展望

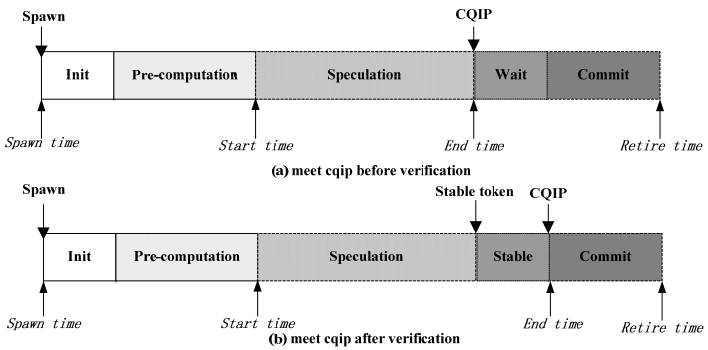
- 随着多核处理器发展，人们与并行的关系密切起来。从技术发展来看，手工并行化所做工作会逐渐转移到自动并行化中，所以这种局面会改变。
- 自动并行化仍有很大发展空间。
  - 适用的并行机模型和并行计算模型还比较单一
  - 存在多种加速比性能瓶颈，缺少理想的解决办法

- 自动并行化与手工并行化的结合有助于开发高效并行应用程序
  - 在二者中间寻找一个结合点
  - 把前者作为后者的后续优化
  - 需要语言革新或创新，这应该是未来并行化的主流方向。
- **SpMT**技术是面向多核处理器自动并行化唯一选择
- 软硬件协同设计仍然是消除**SpMT**自动并行化性能瓶颈的有前景的手段

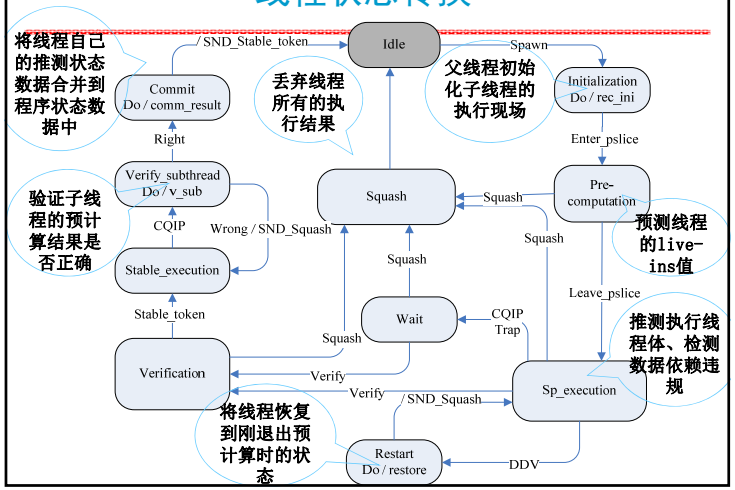
谢谢指正！

附录

线程生命期



线程状态转换



多版本Cache设计

V:有效位

Ver:数据的版本号，用于区分预计算片断与推测线程体中的数据以及推测线程体中发起点前后的数据

O:当产生新的版本时置1

RR:从前驱线程读入数据时置1

M:数据被写入时置1

Addr:数据地址

Value:数据值

Bit	Description
V	Valid
RL	Remote Loaded
M	Modified
Ver	Version
O	Old

(a) cache line state bit

State	V	RL	M	Ver	O
Invalid	0	X	X	X	X
PreSh	1	1	0	0	1
PreEx	1	0	1	0	0
PreExO	1	0	1	0	1
SpSh	1	1	0	TV	0
SpShM	1	1	1	TV	0
SpShO	1	1	1	TV	1
SpEx	1	0	1	TV	0
SpExO	1	0	1	TV	1

(b) state encoding(TV means Thread Version)

V	RL	M	Ver	O	Address Tag	Data

(c) cache line

多版本Cache：寄存器Cache

Bit	Description
V	Valid
L	Loaded First
M	Modified

(a) cache line state bit

State	V	L	M
Init	1	0	0
Validate	1	1	0
MCommit	1	0	1
VaandMC	1	1	1
InValid	0	X	X

(b) state encoding

V	L	M	Tag	Data

(c) cache line

V:数据有效位

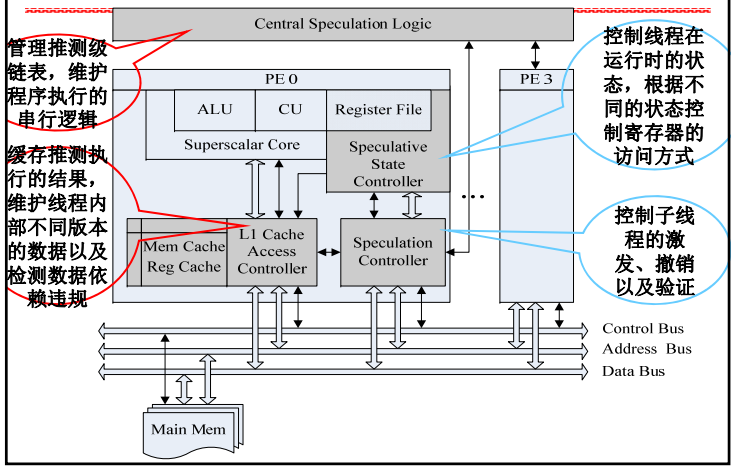
R:读寄存器时被置位

M:写寄存器时被置位

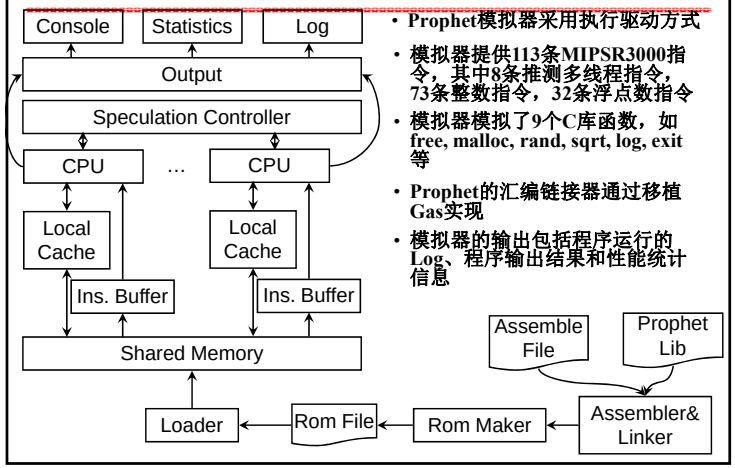
Tag:寄存器标号

Value:寄存器的值

目标机体系结构



Prophet 模拟器实现



- Prophet模拟器采用执行驱动方式
- 模拟器提供113条MIPSR3000指令，其中8条推测多线程指令，73条整数指令，32条浮点数指令
- 模拟器模拟了9个C库函数，如free, malloc, rand, sqrt, log, exit等
- Prophet的汇编链接器通过移植Gas实现
- 模拟器的输出包括程序运行的Log、程序输出结果和性能统计信息

评述

- 在Prophet执行模型中引入了线程预计算，以提高线程推测正确率
- Prophet中使用基于总线监听的机制来解决内存依赖。与多版本cache配合比较完善地解决了内存数据依赖的问题
- 栈模型中的共享栈是当前SpMT技术中解决函数栈问题的基本方案。所提出的独立栈方案比较好的解决了共享栈中栈加锁方案导致多线程执行效率下降问题
- 设计了Prophet体系结构，给出了内存数据依赖的解决方案——多版本cache和寄存器文件
- Prophet体系结构还加入了对超标量流水线的支持，从而使后端模拟器提供了时钟周期级别的模拟精度
- 增加硬件输出缓冲区WOB（Write Out Buffer）解决并行输出问题

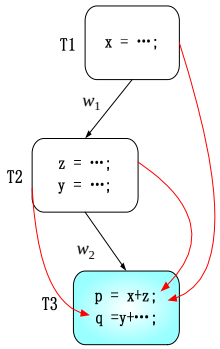
数据依赖定量分析

- 数据依赖弧个数
  - 在一个线程域内，与其他线程有引用（use）依赖关系的弧的个数，

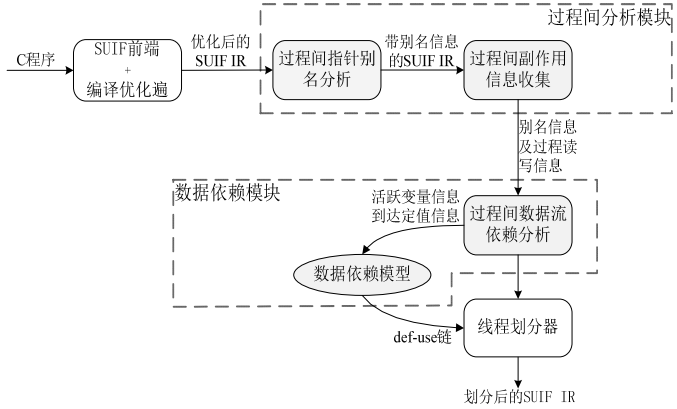
$$Dependence_Count(T) = \sum w_n * A_n$$

- 数据依赖距离

$$Dependence_Distance(T) = \max(dependence_distance(A_i))$$



Prophet数据依赖支持框架



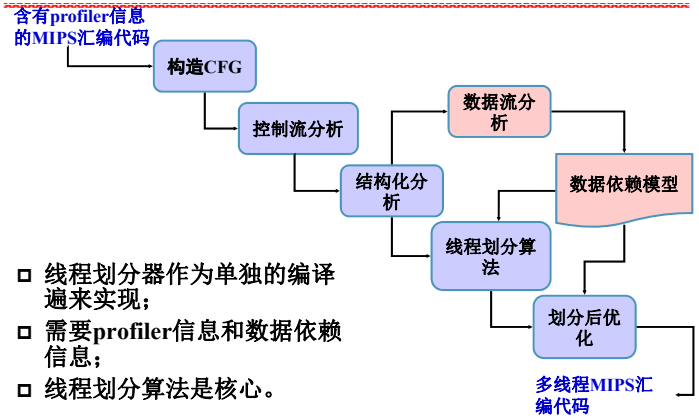
评述

- 依据现有的数据依赖模型DDC和DDD的设计思想，提出了基于SpMT的DDC数据依赖模型方案，能准确反映线程间的数据依赖程度。
- 在父线程中查找子线程中活跃变量的最近有效定值点，作为线程划分算法寻找最近有效依赖弧的判断依据，通过计算候选线程的最优依赖弧的个数，获取满足激发线程的合理位置，指导线程划分。
- 特别地，在线程划分中，为了采用激进的策略，尽可能早地激发后继线程，提出构建预计算片段的方式，根据数据依赖模型提供候选线程的活跃变量以及这些活跃变量的最近有效定值点，实现预计算片段的创建，从而进一步提高了线程划分的效率。

线程划分算法

- 输入
  - 加权控制流图（分支概率、基本块动态指令数、调子指令开销等）；数据依赖模型
- 划分方法
  - 结构化分析，识别循环与非循环，通过建立综合评估控制、数据依赖影响的代价模型，确定最优推测路径；
  - 综合考虑线程粒度、控制依赖、数据依赖、激发距离等因素，沿推测路径选择可以并发的线程以及线程的激发位置输出
  - 插入线程控制指令（SP, CQIP, Squash）的指令流序列，并服从于SpMT执行模型的约束。

Prophet线程划分的一个过程



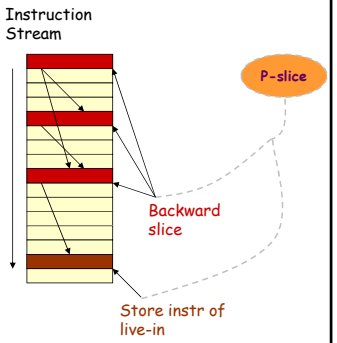
- 线程划分器作为单独的编译遍来实现；
- 需要profiler信息和数据依赖信息；
- 线程划分算法是核心。

预计算片段(p-slice)

p-slice：是该线程的SP点到其起点的这段代码的一个简化版本，由父线程产生的，并可能被子线程用到的数据的指令组成

构造 *p-slice* 的步骤：

- 确定候选线程中 live-ins
- 在当前线程中寻找live-ins的store指令，并后向遍历一定长度，寻找其关联的指令，这些指令作为预计算片段(*p-slice*)
- 将 *p-slice* 作为推测线程的附加部分执行：
  - 推测线程执行时先执行 *p-slice*
  - 由于*p-slice* 很小，不会增加运行时开销

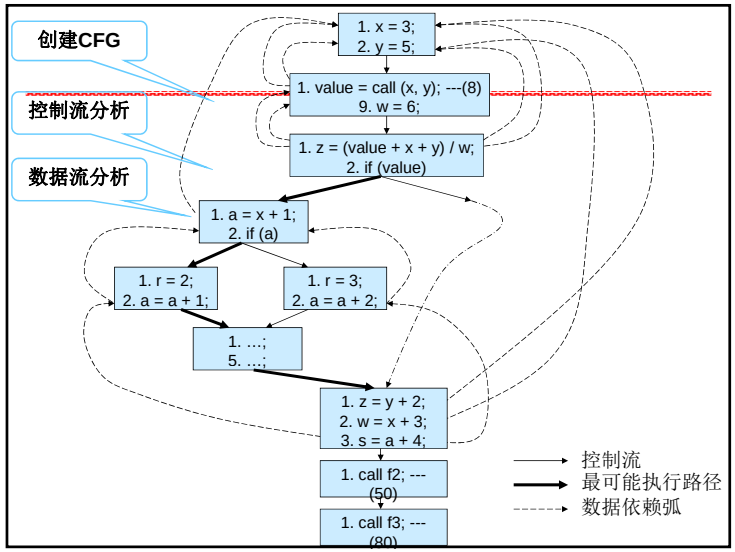


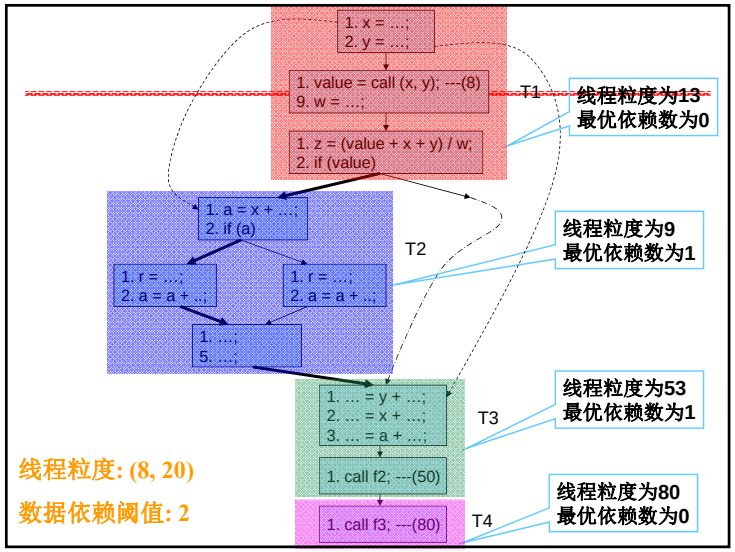
评述

- 实现了三种线程划分方案
- 方案I有平台无关性，并可扩展SUIF的数据流分析库，实现指针分析和过程间分析，使得由数据依赖模型得到的数据依赖信息更加精确和完善；缺点是：只是在虚拟机上运行高级的中间表示，和汇编指令并不是一一对应的，精确度欠佳
- 方案II的优点是：可以应用指针分析和过程间分析提取更多信息（进一步工作）；缺点是：高级中间表示和汇编指令并不是一一对应，所以效率比方案III稍差
- 方案III的优点是：低级中间表示和汇编指令一一对应；缺点是：寄存器分配以过程体为单位，导致线程间存在寄存器依赖；此外线程间还存在相当的内存依赖需要分析和解决；有与目标机相关性。

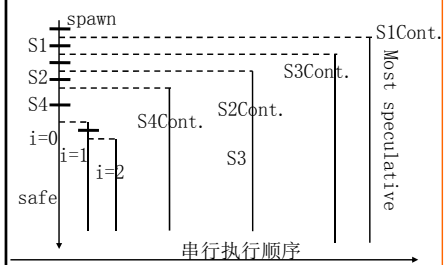
An Example:

```
foo (){
  x = 3;
  y = 5;
  value = f (x, y);
  w = 6;
  z = (value + x + y) / w;
  if (value){
      a = x + 1;
      if (a){
           r = 2;
           a = a + 1;
      }
      else{
          r = 3;
          a = a + 2;
      }
      …;
  }
  else{
    …;
  }
  z = y + 2;
  w = x + 3;
  s = a + 4;
  f2();
  f3();
  return;
}
```





线程激发策略




```
spawn
  S1Cont;
 S1();
 cqip;
S1Cont:
 ...; cqip;
 ...
S1()
{
 ...
 spawn
   S3Cont;
 ...
 spawn
   S2Cont;
 S2();
 cqip;
S2Cont:
 S3();
 cqip;
S3Cont:
 ...; cqip;
```

```
S2()
{
 ...
 spawn
   S4Cont;
 S4();
 cqip;
S4Cont:
 ...
 ...; cqip;
}
S4()
{
 ...
 for(i=0;...)
 {
   spawn it;
   ...
 }
 ...
}
```

Prophet线程状态转换

状态	含义
Idle	空闲状态
Initialization	线程初始化
Pre-promputation	预计算状态
Sp_execution	推测执行状态
Restart	线程重启
Verification	线程接受来自确定线程的验证
Verify_subthread	确定线程验证其子线程
Stable_execution	线程处于确定执行的状态
Commit	线程提交自己的执行结果到内存
Squash	线程被撤销
Wait	线程执行到CQIP点或陷入

动作	含义
rec_ini	接收来自父线程的初始化数据，包括线程起始地址以及其它相关的执行现场
SND_Squash	通知后继子线程撤销
v_sub	确定执行的线程执行到其CQIP点开始验证子线程，验证的值为子线程在预计算中产生的结果
SND_Stable_Token	确定线程通知后继线程以确定方式执行
Comm_result	确定线程将自己的推测执行结果（不包括预计算结果）提交给程序状态
restore	恢复线程的状态到刚刚推出预计时算的状态，线程重新从起始地址开始执行