

文章编号:1001-9081(2010)10-2781-04

基于 TBB 和 Cilk ++ 的并行蚁群算法在路径寻优中的应用

王磊, 曹 茜

(陕西师范大学 计算机科学学院, 西安 710062)

(wanglei_xiaojay@stu.snnu.edu.cn)

摘 要:针对实际道路路网的一类路径寻优问题,提出了带回退机制的蚁群搜索算法,求解在实际道路路网中完成遍历所有规定节点的一条较优路径。为解决大规模实际道路路网数据量大、蚁群算法收敛速度慢的问题,分别采用 Intel Threading Building Blocks(TBB)和 Cilk ++ 并行编程模型实现了并行蚁群搜索。与基于 WinAPI 函数的多线程蚁群算法相比,这两种模型均避免了手动启动线程及识别临界区资源等复杂操作,开发难度降低;在运行效率方面,基于 TBB 的并行蚁群算法和基于 WinAPI 的并行蚁群算法效率接近,而基于 Cilk ++ 的并行蚁群算法在双核环境下,运行效率和加速比都超过了基于 WinAPI 的并行蚁群算法。

关键词:TBB; Cilk ++; 并行蚁群算法; 多核

中图分类号: TP301.6 **文献标志码:** A

Application of parallel ant colony algorithm based on TBB and Cilk ++ in path optimization

WANG Lei, CAO Han

(School of Computer Science, Shaanxi Normal University, Xi'an Shaanxi 710062, China)

Abstract: An Ant Colony Algorithm (ACA) with rollback mechanism was proposed for obtaining optimized path, which traverses all necessary nodes in real road network. In case of the large scale real road network, the convergence speed of traditional sequential ACA was quite slow. Therefore, two parallel ACAs based on Intel Threading Building Blocks (TBB) and Cilk ++ parallel programming model were designed separately. Both of them were easier to operate with simple instructions than complicated thread triggering and critical resource boundary recognition. Therefore, these two models were easier to develop and more applicable compared with WinAPI multi-threaded based parallel ACA (pACA). The experimental results show that pACA based TBB is nearly identical with pACA based on WinAPI in terms of efficiency, while pACA based on Cilk ++ exhibits higher efficiency and better speed up than pACA based on WinAPI in dual-core system.

Key words: Intel Threading Building Blocks (TBB); Cilk ++; parallel Ant Colony Algorithm (ACA); multi-core

0 引言

多核时代的到来,使得传统的串行编程技术面临极大挑战,通过多线程软件技术来提高软件性能已成为目前主要手段。如果熟悉底层平台,可以使用 pthreads 或 WinAPI threads 开发;也可以使用并发开发平台,由平台来自动协调、调度和管理多核资源。并发开发平台包括各种线程池库,如消息传递环境(Message Passing Interface, MPI)、任务编程环境(Intel Threading Building Blocks, TBB)和动态编程环境(Cilk ++、OpenMP)。这些并发平台通过提供语言抽象、扩充注释或者提供库函数的方式来支持多核开发。

目前 OpenMP 及 MPI 应用范围较广,但国内 TBB 及 Cilk ++ 应用文献较少。Lin Chao-Sheng 等人^[1]把 TBB 应用于多核嵌入式软件 VERTAF/Multi-Core (VMC),结果表明 TBB 具有较高加速比,能取得较好并行效果。Li Ni 等人^[2]把 TBB 应用于建模与仿真领域,模拟实验结果表明采用了 TBB 后不仅可以充分利用计算资源,仿真效率也大大提高。Marowka^[3]把 TBB 应用于 π 值求解及 Benchmark 八个应用程序,得出 TBB 在粗粒度编程中具有较好性能。Thulasiram 等人^[4]在 8 节点 SMP 环境下把 Cilk ++ 应用于多线程定价算

法,实验结果表明线程数越少、加速比越大,指出启动较多线程反而会带来许多额外的负载,在处理此类分而治之算法时要综合考虑各种因素,Cilk ++ 是此类算法中最好的平台。

本文在上述理论基础上分析了 TBB 及 Cilk ++ 并行编程模型,把它们应用于蚁群算法来解决西安市路径寻径问题,并与基于 WinAPI 多线程并行蚁群算法进行了比较。

1 Intel TBB 与 Cilk ++ 对比

1.1 TBB 基于任务级编程的优势^[5]

1) 资源匹配。TBB 线程构建模块调度算法使一个逻辑线程映射到一个相应的物理线程,达到最佳运算效率。

2) 任务的启动和停止过程更加迅速。任务和线程相比关键优势是它们更轻量,在 Linux 系统,启动和终止一个任务的速度是线程的 18 倍;在 Windows 系统,这个比率更是超过了 100。

3) 更高效率的优先级评定。线程调度是一种公平的调度方式。TBB 的任务效率高在于它的调度是不公平的,基于任务编程时,任务调度有高级别信息,可以为了效率而牺牲公平性。

4) 提高负载均衡。任务调度会很好地分配任务到线程

收稿日期:2010-04-06;修回日期:2010-05-29。 **基金项目:**陕西师范大学研究生培养创新基金资助项目(2010CXS012)。

作者简介:王磊(1984-),男,陕西咸阳人,硕士研究生,主要研究方向:并行计算;曹茜(1963-),女,陕西西安人,教授,博士,主要研究方向:地理信息系统、并行计算。

中并让各线程负载均衡。

1.2 Cilk++与TBB任务窃取机制的区别

Cilk++采用随机任务窃取机制,当一个硬件线程空闲时,就会随机从其他处理器中窃取任务,窃取的任务位于下一个连续函数地址^[3],如下列代码:

```
void f(void)
{ cilk_spawn A();
  cilk_spawn B(); }
```

执行函数f(),一个线程从函数A()开始执行,另一个空闲的硬件线程就会窃取函数B()。Cilk++的任务窃取机制时间复杂度为 $O(PS)$,其中P是线程个数,S是度。Cilk++的任务窃取方式与TBB不同,它没有窃取函数f(),而是窃取了函数f()未开始运行的子函数B(),文献证明TBB的任务窃取机制在有些情况下效果很差,而Cilk++任务窃取机制具有较好效果^[6]。

1.3 Cilk++性能分析^[7]

引入量(work)和度(span):量指为完成程序所需要的总的运行时间;度指从起点到终点的关键路径。

$T(P)$ 是程序在P个处理器上运行的时间, $T(1)$ 指量, $T(\infty)$ 指度,在两个处理器上, $T(2) \geq T(1)/2$,引入量法则: $T(p) \geq T(1)/p$ 。

相似的,在P个处理器上运行时间大于在无限多的处理器上运行时间,引入度法则: $T(p) \geq T(\infty)$ 。

加速比定义: $speedup = T(1)/T(P)$ 。

并发度定义: $parallelism = T(1)/T(\infty)$ 。

2 蚁群算法及路径寻优问题

2.1 蚁群算法求解旅行商问题

设有m个蚂蚁,n个城市,把这m个蚂蚁随机放在n个城市上($m \leq n$),让这些蚂蚁按式(1)向前移动,并在路径上撒上定量的信息素,直到回到原点,所有的蚂蚁都周游一次后,本次迭代结束。

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}^\alpha(t) \eta_{ij}^\beta(t)}{\sum_{\mu \in Tabu_k} \tau_{i\mu}^\alpha(t) \eta_{i\mu}^\beta(t)}, & j \notin Tabu_k \\ 0, & \text{其他} \end{cases} \quad (1)$$

其中: $p_{ij}^k(t)$ 是t时刻蚂蚁k由i城市到j城市的转移概率; $Tabu_k$ 为蚂蚁k的禁忌表; $\tau_{ij}(t)$ 为t时刻两城市ij之间边上的信息素量; $\eta_{ij}(t)$ 为启发函数,表示t时刻蚂蚁从城市i选择城市j的期望值,旅行商问题(Traveling Salesman Problem, TSP)一般取 $\eta_{ij}(t) = 1/d_{ij}$ (d_{ij} 是ij两城市之间距离); α 表示蚂蚁在运动过程中所积累的信息在蚂蚁选择路径中所起的作用, β 表示启发信息 $\eta_{ij}(t)$ 在蚂蚁选择路径中所起的作用,通常在TSP中 α, β 取值范围为 $[0, 5]$ 。每只蚂蚁周游一次后,根据式(2)进行局部更新。

$$\tau_{ij}(t+n) = \tau_{ij}(t) + \tau_0 \cdot \rho \quad (2)$$

其中: $\tau_0 = C$ (C为常数), $1 - \rho$ 是信息素挥发系数。本次迭代后,进行一次全局更新,把本次迭代后的最优路径按式(3)更新;经过数次迭代后,最终收敛出一条最优路径。

$$\begin{cases} \tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta \tau_{ij}(t, t+n) \\ \Delta \tau_{ij}(t, t+n) = \sum_{k=1}^m \tau_{ij}^k(t, t+n) \end{cases} \quad (3)$$

$\Delta \tau_{ij}^k(t, t+n)$ 是第k只蚂蚁在时间段(t, t+n)留在路径ij上的信息素量, $\Delta \tau_{ij}^k(t, t+n)$ 按式(4)计算:

$$\Delta \tau_{ij}^k(t, t+n) = \begin{cases} Q/L_k, & \text{蚂蚁k经过ij} \\ 0, & \text{其他} \end{cases} \quad (4)$$

其中:Q为常量, L_k 为第k只蚂蚁周游的路径长度。

2.2 路径寻优问题与TSP

本文所指的路径寻优问题类似于经典TSP,在N个节点中取 n ($n \leq N$)个节点,要求遍历完n个节点、回到起点且代价最小,此n个节点不一定两两相通。经典TSP所研究的图是任意两点都相通,但实际道路并不是任两点都相通,用蚁群算法在求解过程中可能存在回溯现象,如图1。

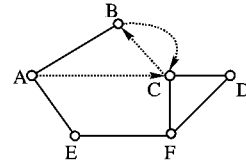


图1 回溯

当蚂蚁k路径为A→C→B时,B节点所有的邻接点都访问过,只能回溯到C点,它走过的路径为A→C→B→C,C点访问两次;而在经典TSP中不存在回溯,且每个节点仅访问一次。

2.3 蚁群算法求解路径寻优流程

1)在西安市地图4525个节点中随机选取n个节点,设定边上的信息素值 τ_{max} ,设 $Q_0 = 0.1$,生成随机数 $Q \in (0, 1)$,把m个蚂蚁放在n个节点上($m \leq n$)。若 $Q \geq Q_0$,算法保证优先访问必经节点,即如果从当前必经节点i到下一个必经节点j道路联通且转移概率相比其他节点高,蚂蚁按式(1)选择下一节点j;如果从当前必经节点i没有一条道路联通下一个必经节点j,蚂蚁选择转移概率相比其他节点较高的下一节点。如果出现图1情况,则选择回溯;若 $Q < Q_0$,每只蚂蚁按式(5)选择下一个节点。

$$\arg \max_{j \in Allowed_k} \{ \tau_{ij}^\alpha(t) \eta_{ij}^\beta(t) \} \quad (5)$$

2)本次迭代后保存最优路径,重置各边上的信息素范围 $[\tau_{min}, \tau_{max}]$,按式(2)进行局部更新。

3)迭代若干次后,判断最优路径在某个时间段是否改变,若改变,继续迭代,按式(3)全局更新;若不改变,迭代结束,保存最优解。

3 基于TBB的并行蚁群算法

3.1 基于TBB的并行蚁群算法原理

在蚁群算法的每次迭代中,蚂蚁群体在选择下一节点时存在着潜在的并行性。传统的算法是一只蚂蚁完成一次周游后,另一只蚂蚁进行下一次周游。用TBB指令后,会把每次迭代中的m个蚂蚁按任务并行,分配给t个线程,这t个线程同时执行。

并行蚁群算法执行代码:

```
class Parallel_Ant
{ void operator()(blocked_range<int> & range)
{ for(i = range.begin(); i != range.end(); ++i)
{ ant.initialAnt(i);
  ant.tbb_putAnt2start(i);
  ant.tbb_putAnt2nextVertex(i);
} } };
void CAnt::tbb_startSearch()
{ parallel_for(blocked_range<int>(0, AntNum), Parallel_Ant(),
  auto_partitioner()); }
```

3.2 基于TBB的并行蚁群算法实验

主机配置如下:CPU为Intel Xeon E5405 2.0 GHz,2 GB内存,Windows XP操作系统,集成开发环境为Visual Studio 2008,并采用C++语言以及Intel C++编译器。蚁群算法参数如下: $\alpha = 1, \beta = 5, \rho = 0.9, \tau_{max} = 1$,蚂蚁数50。把基于TBB并行蚁群算法和基于WinAPI多线程并行蚁群算法进行比较,如图2、3。图中1代表单线程蚁群算法运行结果,raw(2/4)代表用WinAPI启动2/4裸线程运行结果,tbb(2/4)代表基于TBB的并行蚁群算法在双核及四核下运行结果。实

验过程中 CPU 在双核及四核下的使用率记录如表 1。

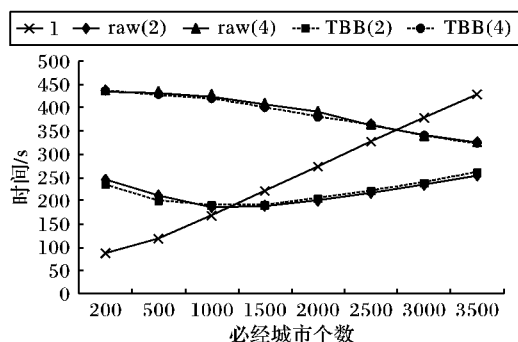


图 2 运行时间比较 1

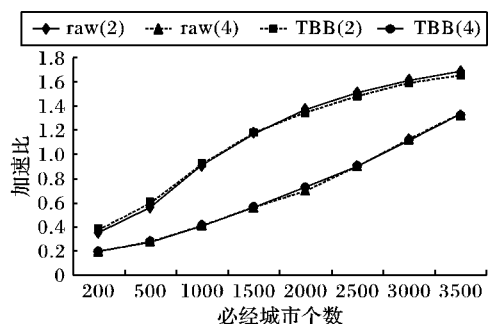


图 3 加速比比较 1

表 1 CPU 使用率记录 1

必经城市数量	CPU	
	双核	四核
200	45% ~ 60%	23% ~ 30%
500	47% ~ 67%	24% ~ 32%
1000	54% ~ 77%	25% ~ 35%
1500	62% ~ 85%	25% ~ 42%
2000	64% ~ 90%	26% ~ 46%
2500	75% ~ 93%	26% ~ 55%
3000	77% ~ 95%	32% ~ 60%
3500	80% ~ 98%	37% ~ 67%

由图 2、3 及表 1 可得出如下结论。

1) 基于 TBB 的并行蚁群算法在双核下运行时间和用 WinAPI 启动 2 个裸线程运行时间相近,在四核下运行时间与用 WinAPI 启动 4 个裸线程运行时间相近。

2) 启动多个线程时,刚开始问题规模较小,线程间切换频繁且初始化多个线程占用较多时间,运行时间均大于串行时间。随着问题规模的扩大,启动多个线程所用时间开始小于串行时间。启动 2 个线程相对启动 4 个线程而言,启动线程个数及线程间切换次数少,故运行时间较早开始少于串行时间。

3) 基于 TBB 的并行蚁群算法与基于 WinAPI 的多线程并行蚁群算法加速比相近且均呈上升趋势。

4) 随着问题规模的扩大,CPU 的使用率变大,多核优势逐渐体现出来。

4 基于 Cilk++ 的并行蚁群算法

4.1 基于 Cilk++ 的并行蚁群算法原理

用 Cilk++ 指令后,把每次迭代中的 m 只蚂蚁按任务并行,分配给 t 个线程,这 t 个线程同时执行。

Cilk++ 并行蚁群算法执行代码:

```
void CAnt::cilk_startSearch()
{ cilk_for(int i=0; i<AntNum; i++)
```

```
{ ant.cilk_putAnt2nextVertex(i); }
extern "Cilk++" void parallel_main(int argc, char *argv[])
{ for(int i=0; i<NcMax; i++)
{ ant.cilk_startSearch();
  ant.cilk_GlobalUpdate(); } }
int main(int argc, char *argv[])
{ return cilk::run(parallel_main, argc, argv); }
```

4.2 基于 Cilk++ 的并行蚁群算法实验

主机配置与 3.2 节相同,编译器采用 Cilk++ 编译器。把基于 Cilk++ 的并行蚁群算法和基于 WinAPI 多线程的并行蚁群算法进行比较,如图 4、5。图中 1 代表单线程蚁群算法运行时间结果,raw(2/4)代表用 WinAPI 启动 2/4 裸线程运行结果,Cilk(2/4)代表基于 Cilk++ 的并行蚁群算法在双核及四核下运行结果。实验过程中 CPU 在双核及四核下的使用率记录如表 2。

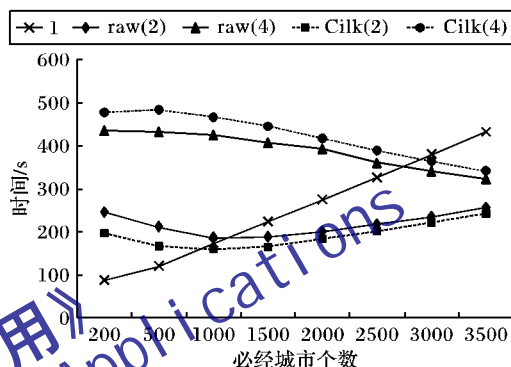


图 4 运行时间比较 2

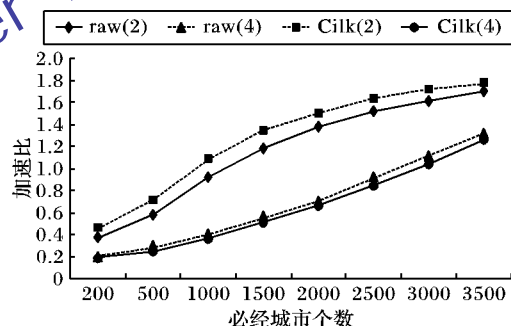


图 5 加速比比较 2

表 2 CPU 使用率记录 2

必经城市数量	CPU	
	双核	四核
200	45% ~ 61%	17% ~ 28%
500	55% ~ 72%	18% ~ 29%
1000	65% ~ 82%	20% ~ 32%
1500	68% ~ 90%	23% ~ 41%
2000	75% ~ 93%	25% ~ 44%
2500	78% ~ 95%	29% ~ 47%
3000	81% ~ 97%	30% ~ 53%
3500	83% ~ 99%	35% ~ 61%

由图 4、5 及表 3 可以得出如下结论。

1) 基于 Cilk++ 的并行蚁群算法在双核下运行时间少于用 WinAPI 启动两个裸线程运行时间,在四核下运行时间大于 WinAPI 启动四个裸线程运行时间。Cilk++ 指令通过不断挖掘程序内部的并行潜力,使得并行程序的开发只需通过简单的 cilk_for、cilk_spawn 与 cilk_sync 来完成,降低了开发难度。在四核下基于 Cilk++ 的并行蚁群算法运行时间多于用 WinAPI 启动 4 个裸线程运行时间,因 Cilk++ 软件开发包本

身存在着开销, Cilk++ 的并行能力借助于运行库的支持, 在程序并行过程中, 这些库也会带来一定的开销, 所以引入较多线程时, 运行效率较低, 但仍然降低了并行开发难度。

2) 启动多个线程, 刚开始问题规模较小, 线程间切换频繁且初始化多个线程占用较多时间, 基于 Cilk++ 的并行蚁群算法及基于 WinAPI 的并行蚁群算法运行时间均大于串行蚁群算法执行时间。随着问题规模的扩大, 启动多个线程运行时间开始小于串行运行时间。启动 2 个线程相对 4 个线程而言, 启动线程个数及线程间切换次数少, 故运行时间较早开始少于串行时间。

3) 基于 Cilk++ 的并行蚁群算法在双核下加速比大于用 WinAPI 启动两个裸线程加速比, 在四核下加速比小于用 WinAPI 启动 4 个裸线程加速比, 且加速比均呈上升趋势。

4) 随着问题规模的扩大, CPU 的使用率变大, 多核优势逐渐体现出来。

4.3 基于 Cilk++ 与 TBB 并行蚁群算法比较

把基于 Cilk++ 的并行蚁群算法和基于 TBB 的并行蚁群算法进行比较, 如图 6、7。图中 1 代表单线程运行结果, TBB (2/4) 代表基于 TBB 的并行蚁群算法在双核及四核下运行结果, Cilk (2/4) 代表基于 Cilk++ 的并行蚁群算法在双核及四核下运行结果。

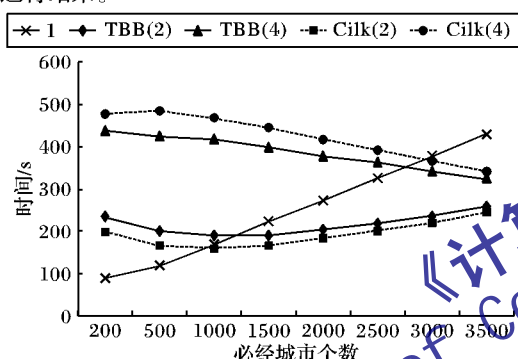


图6 运行时间比较3

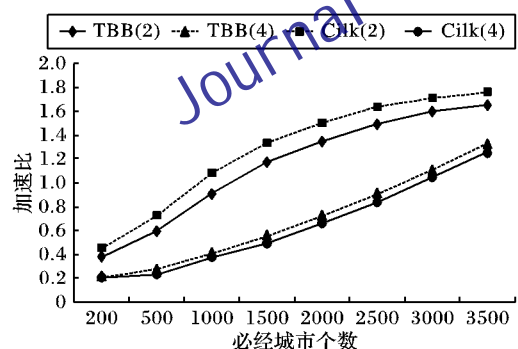


图7 加速比比较3

由图 6、7 可以得出如下结论。

1) 基于 Cilk++ 的并行蚁群算法在双核下运行时间少于基于 TBB 的并行蚁群算法在双核下运行时间, 在四核下运行时间大于后者运行时间。

2) 基于 Cilk++ 的并行蚁群算法在双核下加速比大于基于 TBB 的并行蚁群算法加速比, 在四核下小于后者加速比, 且加速比均呈上升趋势。

5 Amdahl 定律和 Gustafson 定律^[5]

Amdahl 定律: 如果程序中串行部分所占的比率不能得到相同数量级的降低, 那么在获得高并行处理率上所付出的努力将付之东流。

Gustafson 定律: 应该根据处理器数量来增加问题的规模

以获得加速, 而不是将问题的规模保持不变。

Amdahl 定律将程序看做是不变的。Gustafson 指出随着计算机功能强大程序能够完成越来越多的工作, 负载将逐渐增大而不是保持不变。对许多问题来说, 随着问题规模的增长, 问题并行部分工作量增长速度将快于非并行部分。Amdahl 定律和 Gustafson 定律都是正确的, 只不过是看程序在相同负载的情况下运行得更快, 还是希望程序在更大的负载下工作。事实证明, 人们更希望程序变得越来越复杂, 并且解决规模越来越大的问题。Gustafson 定律更符合事实。但 Amdahl 定律仍有很大价值, 它可以使应用程序在相同的测试标准下做研究。

本文研究了不同规模下的基于 Cilk++ 及 TBB 的并行蚁群算法, 实验结果更符合 Gustafson 定律, 即规模越大, 并行效果越明显, 并行计算将有能力为我们解决规模更大、更复杂的问题, 并行计算的应用将得到更大范围的推广。

6 结语

本文采用基于 TBB 和 Cilk++ 的并行蚁群算法解决西安市实际道路路网的路径寻优问题, 并与基于 WinAPI 的多线程蚁群算法进行了比较。实验结果表明, TBB 和 Cilk++ 并行编程模型效果良好。在开发难度方面, 由原来的手动启动线程及识别临界资源等操作转化为只用简单指令即可完成的操作; 在运行效率方面, 基于 TBB 的并行蚁群算法和基于 WinAPI 的并行蚁群算法效率接近, 基于 Cilk++ 的并行蚁群算法在双核环境下加速比甚至超过基于 WinAPI 的并行蚁群算法, 不仅开发难度降低, 而且效率更高。基于 TBB 的并行蚁群算法与基于 Cilk++ 的并行蚁群算法实验对比表明, 前者在四核下运行效率较高, 后者在双核下运行效率较高。这些并行模型为我们编写并行操作带来了极大方便, 而且在运行效率上, 也不低于用 WinAPI 多线程并行模型。

接下来将进一步对 OpenMP、MPI、TBB 及 Cilk++ 几种并行编程模型进行对比分析, 也可将 TBB 和 MPI 相结合, 为多核处理器节点集群提供并行层次化结构, 从而优化集群性能。

参考文献:

- [1] LIN C-S, HSIUNG P-A, LIN S-W, *et al.* VERTAF/Multi-core: A SysML-based application framework for multi-core embedded software development [J]. *Journal of the Chinese Institute of Engineers*, 2009, 32(7): 985-991.
- [2] LI NI, GONG GUANGHONG, PENG XIAOYUAN, *et al.* Scene matching algorithm evaluation based on multi-core parallel computing technology [C]// WCSE 2009: Proceedings of the 2009 WRI World Congress on Software Engineering. Washington, DC: IEEE Computer Society, 2009: 94-98.
- [3] MAROWKA A. Towards high-level parallel programming models for multicore systems [C]// ASE 2008: Proceedings of the 2008 Advanced Software Engineering and Its Applications. Washington, DC: IEEE Computer Society, 2008: 226-229.
- [4] THULASIRAM R K, THULASIRAMAN P, AKIELE C, *et al.* Performance analysis of a multithreaded pricing algorithm on cilk [C]// HPCS02: Proceedings of the 16th Annual International Symposium on High Performance Computing Systems and Applications. Washington, DC: IEEE Computer Society, 2002: 1-7.
- [5] REINDERS J. Intel threading building blocks [M]. [S.l.]: O'Reilly Media, 2007.
- [6] SUKHA J. Brief Announcement: A lower bound for depth-restricted work stealing [C]// SPAA 2009: Proceedings of the 21st Annual Symposium on Parallelism in Algorithms and Architectures. New York: ACM, 2009: 124-126.
- [7] Intel Cilk++ SDK programmer's guide [S/OL]. [2010-01-02]. http://www.owl.net.rice.edu/~comp422/resources/Intel_Cilk++_Programmers_Guide.pdf.