计算任务与体系结构匹配的异构计算可扩展性分析

郝水侠[1,2,3],曾国荪[1,2],谭一鸣[1,2]
(1.同济大学计算机科学及技术系,上海 201804; 2.国家高性能计算机工程技术中心同济分中心,上海 201804;
3.徐州师范大学数学科学学院,江苏徐州 221116)

摘 要: 扩展性是衡量高性能并行系统的一个关键要素,而扩展性的研究主要集中在同构的高性能系统上,异构系统研究的文献很少. 本文以异构系统为研究对象,根据实际应用任务,将计算任务分为三类:单任务模型、元任务池模型和 fork-join 任务队列模型,并给出这三类计算任务的定义. 提出描述基于计算任务和体系结构相匹配的异构计算系统匹配矩阵,给出异构计算的可扩展性定义. 针对上述三种计算任务模型以及异构匹配给出异构系统的可扩展性条件,为异构系统的可扩展性提供了理论依据. 用实例分析证实了这种方法的有效性.

关键词: 异构计算; 匹配矩阵; 可扩展性条件
中图分类号: TP338 **文献标识码:** A **文章编号:** 0372-2112 (2010) 11-2585-05

Scalability Analysis of Heterogeneous Computing Based on Computation Task and Architecture to Match

HAO Shui-xia[1,2,3], ZENG Guo-sun[1,2], TAN Yi-ming[1,2]
(1. *Department of Computer Science and Technology, Tongji University, Shanghai* 201804, *China*;
2. *Tongji Branch, National Engineering & Technology Center of High Performance Computer, Shanghai* 201804, *China*;
3 *School of Mathematical Science, Xuzhou Normal University, Xuzhou, Jiangsu* 221116, *China*)

Abstract: Scalability is a key factor of weighing up high performance parallel system. However, conventional scalabilities research is mainly focused on homogeneous high performance parallel system, heterogeneous parallel is relatively small. The paper which based on heterogeneous parallel system, is sorted computing task to single task model, meta task pool model and fork-join task queue model, and given their definition, their scalability condition on advancing heterogeneous matching matrix. Finally, sample analysis have verified the analytical results and confirmed that the proposed scalability works well in high performance heterogeneous parallel system.

Key words: heterogeneous computing; matching matrix; scalability conditions

1 引言

通常,建立大型并行系统是通过增加系统规模来实现的,因此,研究系统规模变化对并行系统和并行算法影响特别重要,并行系统可扩展性是衡量系统是否能够进行扩展的一个重要指标[8].可扩展性也是并行系统性能评估和优化的基本因素[1,3].

目前在可扩展性研究中,大部分集中在同构系统中,如 Deshpande 在 1986 年提出体系结构可扩展性,Bell 在 1992 年定性描述了并行机的可扩展性[8]. Grama 在 1993 年给出了并行计算的等效率可扩展性模型[3],Sun 在 1994 年提出了等速度可扩展性模型[4].迟利华等提出等并行开销计算比可扩展模型[5].在异构系统可扩展性研究中,如 Sun 在 2002[1]年研究了两个可扩展系统执行速度的比较,陈勇[5]、Sun[7]用标志速度来描述异构系统的性能,比用机器个数描述异构系统有了很大改进.但在异构系统中,还有一个重要因素是计算任务和体系结构相匹配的问题,只有两者匹配时异构系统的性能才可能达到高效.因此本文提出用计算任务和体系结构相匹配的匹配矩阵来表示系统性能,并对计算任务进行分类,给出异构计算系统等速度可扩展函数.

收稿日期:2010-03-19;修回日期:2010-06-14
基金项目:国家 863 高技术研究发展计划(No.2007AA01Z425,No.2009AA012201);国家 973 重点基础研究发展规划(No.2007CB316502);国家自然基金(No.90718015);NSFC-微软亚洲研究院联合资助项目(No.60970155);教育部博士点基金(No.20090072110035);上海市优秀学科带头人计划(No.10XD1404400);高效能服务器和存储技术国家重点实验室开放基金(No.2009HSSA06)

2 异构计算模型

所谓异构计算是指将性能各异的计算机(如:PC、工作站群、向量机、SIMD、MIMD 计算机、FPGA、GPU、DSP、CELL 专用机等),通过高速网络连成并行计算环境,充分利用程序和结构的异构性,各尽潜能,合理分治,协同计算一个应用任务,使得完成时间最小的过程[9].

2.1 体系结构模型

体系结构模型是指抽象的描述计算系统的组成部分.因此异构计算体系结构模型可抽象为 $\boldsymbol{H}_n=\{h_1,h_2,\cdots,h_n\}$,其中 h_i 为单个异构处理器,n 代表为体系结构中处理器的个数.

2.2 计算任务模型

在解决实际问题时将各种并行计算任务的基本特征抽象出来,形成抽象的计算任务模型,可将所有的并行算法抽象成三种情况:单任务模型、元任务池模型和 fork-join 任务队列模型.分别如图 1、图 2、图 3.

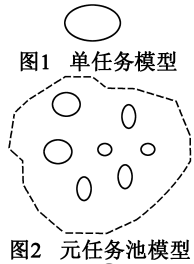


图1 单任务模型

图2 元任务池模型

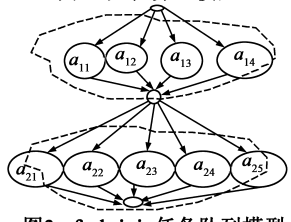


图3 fork-join任务队列模型

(1)单任务模型 单任务通常是指功能相对独立的一段应用程序.本文所指的是不可再分的应用程序,即元任务.特别对于一个复杂的大型应用问题,为了简化编程,省略子任务划分步骤,可以将整个应用程序看成是一个粗粒度的单任务.

(2)元任务池模型 元任务池通常指由功能相对独立的若干个元任务组成一个任务组,如图 2.任务池中的元任务由若干个没有直接关联的子任务构成.可以将元任务池的计算任务用一维数组 $V=\{a_1,a_2,\cdots,a_m\}$ 来表示,a_i 表示第 i 个元任务,当元任务池有 m 个任务,则最多由 m 个机器并发执行.

(3)fork-join 任务队列模型 在实际的高性能计算中,计算任务的子任务间往往有很复杂的关系,许多问题都可以抽象为 fork-join 任务图.因此我们将这种 fork-join 任务图抽象成 fork-join 任务队列模型,如图 3.许多实际问题都可以被抽象成 fork-join 任务队列.如 HPF 中的 doall 结构,C/C ++ 中的 par 和 Parfor 结构,Multi-PASCAL 中的 cobegin 和 coend 等等.这种任务队列模型表达能力最强.

这三种模型既相互独立又有一定的联系,(2)中的 $m=1$ 时转化成(1),故(1)是(2)的特例.(2)又是(3)的特例,每个 join 部分都可以看成是一个元任务池.

2.3 计算任务与体系结构的匹配矩阵

m 个计算子任务分配到由 n 个不同类型的处理器的异构计算系统,有多种的分配方法.不同的分配方法性能不同,甚至差距很大.因此在异构计算系统中,特别要考虑计算任务模式和体系结构类型匹配的情况.为了表达和刻画这种匹配程度,我们提出计算任务 $\boldsymbol{Am}=\{a_1,a_2,\cdots,a_m\}$ 和体系结构 $\boldsymbol{H}_n=\{h_1,h_2,\cdots,h_n\}$ 的匹配矩阵 $\boldsymbol{MV}=(v_{ij})_{m\times n}$,其中 $i\in[1,m]$,$j\in[1,n]$,如图 5.其中 v_{ij} 是计算任务 a_i 在处理器上 h_j 上的相对执行速度.一般情况下,如果 $i\neq x$,$j\neq y$,则 $v_{ij}\neq v_{xy}$.如果匹配矩阵中同行中每个元素都相等,则转化成同构计算系统情况.

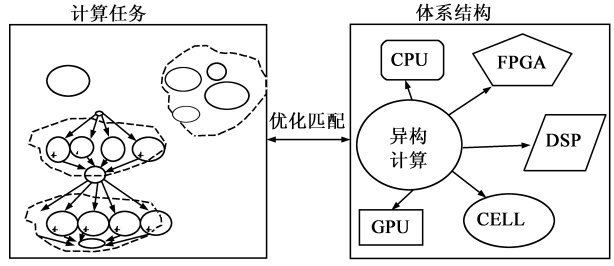


图4 计算任务和体系结构匹配

$$\begin{array}{c} \\ a_1 \\ a_2 \\ \cdots \\ a_m \end{array}\begin{array}{c} \begin{array}{cccc} h_1 & h_2 & \cdots & h_n \end{array} \\ \begin{pmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ v_{m1} & v_{m2} & \cdots & v_{mn} \end{pmatrix} \end{array}$$

图 5 计算任务和体系结构的匹配矩阵

3 异构系统的可扩展性分析

可扩展性是设计并行算法和高性能并行计算机所追求的一个重要目标,也是异构系统所追求的目标.传统意义上可扩展性是指在确定的应用背景下,显示并行系统能力与处理机数目成线形比例的特性,一般指计算机系统(程序)性能随处理器数目的增加而按比例提高的能力[9].

3.1 并行计算系统的可扩展性定义

定义 1(工作负载 W) 有时称问题规模,指程序中计算操作的数目,或求解问题的工作量.理论上是问题输入集规模 n 的函数(n 的多项式、对数、指数函数等).

定义 2(等效率可扩展性函数)[8] 设 P,P'分别是并行系统扩展前后中处理器的个数,$Sp(W)$、$Sp'(W')$是并行系统扩展前后的加速比,则记 $\Psi(p,p')=\dfrac{S_p(W)/p}{S_{p'}(W')/p'}=\dfrac{Ep}{E_{p'}}$ 为等效率可扩展性函数.

$\Psi(p,p')$—理想值为 1,实际值 <1,但希望接近常数.

定义 3(等速度可扩展的)[8]　在并行计算系统中,对于运行在处理器上的某个算法或任务,当处理器数目增大时,若增大一定的工作量能维持整个并行计算系统的平均速度不变,则称并行系统为等速度可扩展的.

定义 4(等速度可扩展性函数)　在并行计算系统中,设 $\bar{v}$ 为系统处理器个数为 P 时的平均速度,W 为计算任务的 W 工作负载;设 $\bar{v}'$ 系统规模扩展后处理器个数为 P' 时的平均速度,W' 为计算任务的工作负载,则记 $\Psi_v(P,P')=\frac{\bar{v}}{\bar{v}'}$ 为等速度可扩展性函数.特别在同构并行计算系统中,$\Psi_v(P,P')=\frac{W/P}{W'/P'}=\frac{WP'}{W'P}$ 为等速度可扩展性函数[5].

用于上述公式计算出的值介于 0 与 1 之间,值越大表示可扩展性越好.显然 $\Psi_v(P,P')=1$,说明平均速度为常数,并行系统是等速度可扩展的.

定义 2 和定义 4 用机器的个数来表示系统的整体性能.在异构系统中,每个机器的性能不相同,仅用机器个数表示异构系统的大小是不适合的.Sun 在文献[7]用标志速度代表系统的大小,它更能反映出异构系统的特征,代表了系统的能力.但在实际应用中,异构系统的性能在很大的程度上取决于算法和体系结构是否匹配.因此,本节利用上节中给出的异构系统的匹配矩阵描述计算任务和体系结构相匹配的异构系统的性能,并企图挖掘异构系统的潜能,度量系统的可扩展性.

3.2　异构计算系统的可扩展性定义

定义 5(异构系统等速度可扩展的)　在异构计算系统中,对于运行在处理器上的某个算法或任务,当处理器数目增大时,若增大一定的工作量能维持整个异构系统的平均速度不变,则称异构系统为等速度可扩展的.

3.3　异构计算系统的可扩展性条件

3.3.1　单任务模型可扩展性条件

在单任务模型中,只有一个计算任务 t_1,匹配矩阵退化为一行向量:$(v_{11},v_{12},\cdots,v_{1n})$,其中 $v_{1i}(i=1,2,\cdots,n)$代表 t_1 任务在不同处理器上的执行速度.假设单任务的工作负载扩展前后分别为 W、W',异构系统处理器规模扩展前后分别为 P、P',匹配矩阵扩展前后分别为$(v_{11},v_{12},\cdots,v_{1P})$、$(v_{11},v_{12},\cdots,v_{1P'})$,则根据异构计算优化调度理论,单任务必然在处理速度最快的机器上执行,其它处理器处于空闲状态.所以单任务 t_1 完成时间在扩展前后分别为$\frac{W}{\max\ v_{1i}}$, $i\in[1,P]$, $\frac{W'}{\max\ v_{1j}}$, $j\in[1,P']$.因此,根据定义 3,等速度可扩展性函数为:

$$\Psi_v(P,P')=\frac{\max(v_{11},v_{12},\cdots,v_{1P})}{\max(v_{11},v_{12},\cdots,v_{1P'})}=\frac{\max\limits_{i\in[1,P]}v_{1i}}{\max\limits_{j\in[1,P']}v_{1j}}.$$

从上式可以看出,在单任务模型中,如果要求等速度可扩展的,则要求计算系统扩展后的某个处理器的速度等于扩展前所有处理器中最快的一个处理器速度,扩展新添的处理器是已有的处理器数量上的重复.显然这样的等速度扩展没有实际意义,所以单任务一般不强求可扩展性.

3.3.2　元任务池模型可扩展性条件

在元任务池模型中,存在 m 个无相关性的子任务集$\{a_1,a_2,\cdots,a_m\}$,设每个子任务的工作负载为 w_i, $i\in[1,m]$,子任务和处理系统的匹配矩阵为 $\boldsymbol{MV}=(v_{ij})_{m\times n}$,则整个元任务池的总负载为 $W=\sum_{i=1}^{m}w_i$.记 T 为子任务完成时间,则异构系统执行元任务池的总体平均速度 $\bar{V}=\frac{W}{T}$.因为元任务不可再分,故规定它只在一个处理器上执行,这符合实际做法.为了讨论扩展性条件,我们假设 $m>n$,即子任务数大于处理器的数量,反之讨论系统扩展性变得没有意义,因为扩展增加的处理器仍然处于空闲,没有子任务需要执行.因此,执行完成时间 T,可由以下几种情况计算得到:

(1) $m\geqslant n$, $n=1$, $T=\frac{w_1}{v_1}+\frac{w_2}{v_1}+\cdots\cdots+\frac{w_m}{v_1}$;

(2) $m\geqslant n$, $n=m$, $T=\max\limits_{1\leqslant i\leqslant m}\{\frac{w_i}{\max\limits_{j\in\Omega(i)}\{v_{ij}\}}\}$,$\Omega((i)$表示当前还没有获得任务的剩余机器.

(3) $m>n$,一般情况下,T 可由下面算法计算获得.

算法 Calculating_Final_Time

输入:每个子任务的工作负载 w_i, $i\in[1,m]$和匹配矩阵 $\boldsymbol{MV}=(v_{ij})_{m\times n}$

输出:total_task_execution_time

```
{  L←({a1, a2, …, am}};      //元任务池队列
   ρ←{h1, h2, …, hn};        //空闲处理器集合
   ε←∅;                      //子任务执行队列初始化为空
   ts(ai)=0, te(ai)=0, 1≤i≤m;     //所有子任务的开始和结束执行时间初始化为0
   ts(hj)=0, te(hj)=0, 1≤j≤n;     //所有处理器的开始和结束执行时间初始化为0
   do until L=∅
   {  for each idle processor h∈ρ
      (ai, hj)←select_a_p(min{wi/vij, …});//选择运行时间最短的处理器和子任务配对(ai, hj)
      te(ai) = max(ts(ai) + wi/vij, ts(hj) + wi/vij);
      ε←ε + {ai};
      ρ←ρ - {hj};                 //将正在处理任务的处理器从空闲队列中去除
      L←L - {ai};
   for each executable task ak∈ε
```

$h_j \leftarrow$processor(a_k); //求执行子任务 a_k 的处理器

$t_s(h_j) = t_e(a_k)$;

$\varepsilon \leftarrow \varepsilon - \{a_k\}$;

$\rho \leftarrow \rho + \{h_j\}$; //将已处理完任务的处理器加入空闲队列中

}

total_task_execution_time($\max\{t_e(i), 1 \leqslant i \leqslant m\}$);

}

假设元任务池的工作负载扩展前后分别为 W、W'，异构系统处理器规模扩展前后分别为 n、n'，匹配矩阵扩展前后分别为 $\boldsymbol{MV} = (v_{ij})_{m \times n}$，$\boldsymbol{MV}' = (v'_{ij})_{m \times n'}$.

研究可扩展性的目的之一是设计应用子任务的精度和负载，配置计算系统的结构和处理能力，合理和优化使用计算资源. 对于元任务池模型，只有扩展后 $n' > n$ 且 $n' = m$ 时最有意义. 当 $n' > m$，实质上仍可转换成 $n' = m$ 的情况，所以我们重点讨论扩展后 $n' > n$ 且 $n' = m$ 时的情况. 此时假设元任务池在扩展前后的完成时间分别为 T、T'，则元任务池模型速度可扩展性函数为 $\Psi_v(P, P') = \frac{\bar{v}}{\bar{v}'} = \frac{W}{T} / (\frac{W'}{T'}) = \frac{WT'}{W'T}$. 为了保持等速度可扩展，相应的元任务工作负载 w'_i 需要扩展，即要求 $(w_1 + w_2 + \cdots + w_m)T' = (w'_1 + w'_2 + \cdots + w'_m)T$. 用上述公式求 w'_i 有无穷多解，但至少存在一组可行解，如下：

$w'_i = \frac{T'}{T} w_i, i([1, m]$. 其中 T、T'是 w_i 和 v_{ij}的函数，由上面(1)(2)(3)来计算.

3.3.3 fork-join 任务模型可扩展性分析

典型的 fork-join 任务模型如图 3，因为在计算任务中总是串中有并，并中有串. 所以此模型更好地描述了实际应用任务的情况. 为了分析方便，将每个 fork 层都看成是一个元任务池，相应的任务记为 $a_1^i, a_2^i, \cdots, a_j^i, \cdots, a_{l(i)}^i$. 其中 i 代表第 i 个 fork 层；a_j^i 代表第 i 个 fork 层的第 j 个任务；$l(i)$ 代表每第 i 层的任务总数. 在每个 join 点，是前面 fork 层所有子任务的一个同步，为了描述方便，可以假设这个节点的任务负载非常小忽略不计. 若 fork-join 任务队列中总的 fork 层为 l，所有的 fork-join 任务队列则形成 l 层的任务队列池，每一层的任务总数分别记为 $l(1), l(2), \cdots, l(l)$. 如图 6，其相应的负载队列如图 7.

$$\begin{matrix} w_1^1 & w_2^1 & \cdots & w_{l(1)}^1 \\ w_1^2 & w_2^2 & \cdots & w_{l(2)}^2 \\ \cdots & \cdots & \cdots & \cdots \\ w_1^l & w_2^l & \cdots & w_{l(l)}^l \end{matrix}$$

图 6 fork-join 的任务队列

$$\begin{matrix} a_1^1 & a_2^1 & \cdots & a_{l(1)}^1 \\ a_1^2 & a_2^2 & \cdots & a_{l(2)}^2 \\ \cdots & \cdots & \cdots & \cdots \\ a_1^l & a_2^l & \cdots & a_{l(l)}^l \end{matrix}$$

图 7 fork-join 任务队列相应的任务负载量

依照上面的定义，总工作负载为：

$$W = \sum_{i=1}^{l(1)} w_i^1 + \sum_{i=1}^{l(2)} w_i^2 + \cdots + \sum_{i=1}^{l(l)} w_i^l = \sum_{i=1}^{l} \sum_{j=1}^{l(i)} w_j^i$$

设总执行时间为 T，则 $\bar{V} = W/T$，其中 T 由下面方法计算： $T = T^1 + T^2 + \cdots + T^l$

$T^1, T^2, \cdots, T^l$ 分别代表第 i 个 fork 层总的执行时间，其值可以根据 3.3.2 中的方法计算.

现假设 fork-join 任务扩展前后的工作负载分别为 W、W'，异构系统处理器规模扩展前后分别为 n、n'，匹配矩阵扩展前后分别为 $MV = (v_{ij})_{m \times n}$，$MV' = (v'_{ij})_{m \times n'}$. 完成时间分别为 T，T'，则 fork-join 任务队列的速度可扩展性函数为：

$$\Psi_v(P, P') = \frac{\bar{v}}{\bar{v}'} = \frac{W}{T} / (\frac{W'}{T'}) = \frac{WT'}{W'T}$$

为了保持等速度可扩展的，相应的元任务工作负载 w'^i_j 需要扩展，即需要满足：

$$\sum_{i=1}^{l} \sum_{j=1}^{l(i)} w_j^i T' = \sum_{i=1}^{l} \sum_{j=1}^{l(i)} w'^i_j T$$

用上述公式求 w'^i_j 有无穷多解，但至少存在一组可行解，如下： $w'^i_j = \frac{T'}{T} w_j^i, i \in [1, l], j \in [1, l(i)]$

特别：对于等速度要求扩展后时间 T' 的计算，可以有多种方法. 一种可行的方法是：假设计算任务规模不变，异构系统规模扩展了，即已知 fork-join 任务扩展“前”的工作负载 W，以及异构系统处理器规模扩展“后”匹配矩阵 $\boldsymbol{MV}' = (v'_{ij})_{m \times n'}$，求出在这样一种环境下，“老”任务在“新”体系结构下的完成时间 T_f，将 T_f 看做是 T'. 显然 $T_f \leqslant T'$，T_f 是 T' 的下界. 另一种合理的计算方法是：估算出扩展后可接受的任务完成时间的上界.

4 实例分析

本实验以文献[10]中的可扩展异构计算 SHOC 基准测试工具集为实验平台，其测试工具集来源于 http://ft.ornl.gov/~kspafford/shoc-0.8.2.tgz. 首先在 linux 环境下安装了 MPICH，CUDA 和 OPENCL，然后再安装基准测试软件. 在这种环境中可以实现将多处理器，GPU，FPGA 等不同部件的异构环境. 另外该基准测试可以进行压力测试和性能测试，其中性能测试又进一步细分为串行测试、伪并行测试(无通信的并行测试)和真并行测试，每种测试又分为 0 级测试，1 级测试和稳定性测试. 根据本文可扩展性分析的需要，我们利用 SHOC 性能测试中的 1 级测试对 2D-FFT，矩阵相乘和逆 2D-FFT 等程序进行了三组实验，如下页表 1 是在等速度可扩展的异构系统中，变化不同的问题规模，得出不同的运行时间，实验结果验证了前面提出的可扩展条件.

表 1　图像处理中各个任务 SHOC 的实验数据

实验编号	子任务	扩展前任务规模和处理速度			扩展后任务规模和处理速度				
		问题规模	联想 PC	联想 PC + 1 块 GPU	问题规模	联想 PC	联想 PC + 1 块 GPU	联想 PC + 1 块 FPGA	曙光 3000
(1)	2D-FFT	10000	1.24	1.88	11000	1.24	1.88	1.89	2.10
	2D-FFT	20000	1.21	1.84	22000	1.21	1.84	1.88	2.09
	M × M	30000	1.05	1.49	33000	1.05	1.49	1.50	2.58
	Inv-2D-FFT	40000	1.04	1.50	44000	1.06	1.50	1.36	2.32
	任务完成时间	0.032s			0.0350s				
(2)	2D-FFT	40000	1.18	1.80	44000	1.18	1.80	1.82	2.10
	2D-FFT	80000	1.15	1.70	88000	1.15	1.70	1.73	2.03
	M × M	120000	1.01	1.48	132000	1.01	1.48	1.44	1.92
	Inv-2D-FFT	160000	1.03	1.46	176000	1.03	1.46	1.40	1.88
	任务完成时间	0.134s			0.148s				
(3)	2D-FFT	160000	1.14	1.81	176000	1.14	1.81	1.83	2.04
	2D-FFT	320000	1.10	1.64	352000	1.10	1.64	1.68	1.88
	M × M	480000	1.04	1.51	528000	1.04	1.51	1.55	1.90
	Inv-2D-FFT	640000	1.00	1.48	704000	1.00	1.48	1.50	1.80
	任务完成时间	0.655s			0.699s				

5　结束语

本论文建立了计算任务和体系结构相匹配的匹配矩阵以及三种并行计算模型,提出异构计算系统可扩展性条件及保持等速度可扩展问题规模的计算方法.该方法比同构计算方法更具有普遍适用性,与文献中列举的相关研究相比更为深入,并且进行了扩展,具有一定的创新和贡献.论文提出的三种计算模型的等速度可扩展性条件为异构系统的可扩展提供了理论依据和方法,丰富了异构系统可扩展性研究内容,为现代高性能系统的研究进一步深入提供了新的思想.

参考文献:

[1] X. Sun. Scalability versus execution time in scalable system [J]. Journal of Parallel and Distributed Computing, 2002, 62 (2):173 – 192.

[2] 谭明锋,龚正虎.基于 ASIC 实现的高速可扩展并行 IP 路由查找算法[J].电子学报,2005,33(2):209 – 213.
Tan Ming-feng, Gong Zheng-hu. High speed IP lookup algorithm with scalability and parallelism based on ASIC implementation[J]. Acta Electronica Sinica, 2005, 33(2):209 – 213. (in Chinese)

[3] A. Gramma, A. Gupta, V. Kumar. Isoefficiensy function: a scalability metic for parallel algorithms and architecures[J]. IEEE Parallel and Distributed Technology, 1993, 1(3):12 – 21.

[4] X. Sun, D. Rover, Scalability of parallel algorithm-machine combinations[J]. IEEE Transaction on Parallel Distributed Systems, 1994, 5:599 – 613.

[5] 迟利华,刘杰,李晓梅,等.并行算法与并行机相结合的可扩展性[J].计算机研究与发展,1999,36(1):47 – 51.
Chi li-hua, Liu jie, Li xiao-mei , etc. Scalability of parallel algorithm combined with parallel machine[J]. Journal of Computer Research and Development, 1999, 36(1): 47 – 51. (in Chinese)

[6] Y Chen, X Sun, M Wu. Algorithm – system scalability of heterogeneous computing[J]. Journal of Parallel and Distributed Computing, 2008, 68(11):1403 – 1412.

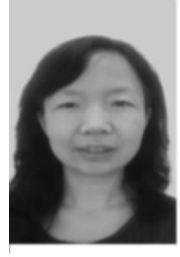
[7] X. Sun, Y. Chen, M. Wu. Scalability of heterogeneous computing[A]. Proceedings of the 34th International Conference on Parallel Processing[C]. Los A lamitos: The IEEE Computer Society, 2005. 557 – 564.

[8] 李晓梅,莫则尧,胡庆丰,等.可扩展并行算法的设计和分析[M].长沙:国防工业出版社,2000. 42 – 46.

[9] 曾国荪,陆鑫达.异构计算中的负载共享[J].软件学报,2000,11 (4):551 – 557.
Zeng guo-sun, Lu xin-da. Load sharing in heterogeneous computing[J]. Journal of Software, 2000, 11 (4): 551 – 557. (in Chinese).

[10] A Danalis, G Marin, C Mccurdy, et al. The scalable heterogeneous computing (SHOC) benchmark suite[A]. Proceedings of 10th GPGPU[C]. New York: ACM, 2010. 1 – 12.

作者简介:



郝水侠　女,1973 年生于陕西大荔县,同济大学博士研究生,研究方向为异构计算、重构计算.
E-mail:sxhao@xznu.edu.cn

曾国荪　男,1964 年生于江西,博士,教授,博导,研究方向为异构计算、信息安全.