基于CUDA实现MRRR算法并行

汪丽杰 赵永华

(中国科学院计算机网络信息中心超算中心 北京 100190)

摘 要 MRRR(Multiple Relatively Robust Representations)算法是求解对称三对角矩阵本征值问题高效、精确的算法之一。在分析MRRR算法及CUDA(Compute Unified Device Architecture)并行体系结构的基础上,针对算法的可并行性,采用单指令多线程并行方式实现了基于CUDA的MRRR算法并行,并从存储结构方面优化算法。实验结果显示,与LAPACK库中串行MRRR实现相比,并行方法在保证精度的基础上获得了20倍的加速比,进而从计算精度和计算时间上说明MRRR算法适合在GPU上并行。

关键词 MRRR,并行,CUDA,本征问题

Parallel Realization of the MRRR Algorithm Based on CUDA

WANG Li-jie ZHAO Yong-hua

(Department of Supercomputing Center, Chinese Academy of Sciences, Beijing 100190, China)

Abstract The algorithm of multiple relatively robust representations(MRRR) is one of the fastest and most accurate algorithms. After analyzing the MRRR algorithm and CUDA parallel architecture, parallel MRRR algorithm based on CUDA was given, and explored the optimization in memory structure. Compared with LAPACK's MRRR implementation this parallel method provides 20-fold speedups. This result demonstrates the algorithm can be mapped efficiently onto GPU.

Keywords MRRR, Parallel, CUDA, Eigenproblem

1 引言

对称矩阵本征值问题 $A\mu_i=\lambda_i\mu_i$ 在计算科学领域普遍存在,包括量子化学、电子结构、结构力学、纳米材料、计算物理、模态识别、自动化控制等,其中 $A\in R^{n*n}$ 是对称矩阵,$\lambda_i\in R$ 为本征值,$\mu_i\in R^n$ 是本征向量。对于大部分应用,计算时间和存储空间已成为解决问题的限制因素,因此需要一种能够高效执行的并行方法。计算对称矩阵本征对的方法通常分3步,第一步是把给定的对称矩阵 A 转化成一个三对角矩阵 T,第二步求解三对角矩阵的本征值和本征向量,第三步是把 T 的本征向量映射回 A。在计算过程中求解三对角矩阵的本征对占时较长,成为整个问题求解的瓶颈之一。很多人提出了解决三对角矩阵本征问题的算法,主要有QR方法、Divide & Conquer算法以及MRRR算法等。

MRRR算法是Dhillon和Parlett在20世纪90年代提出的一种求解对称三对角矩阵本征值问题的高效算法[2]。该方法避免显式正交化使得求解全部本征值和本征向量的时间复杂度为 $O(n^2)$,并且可以保证计算结果的精确和所求解本征向量的正交性。在此之前,求解对称三对角矩阵本征值问题的方法通常需要使用Gram-Schmidt算法或者一个相似的技术对所求解的本征向量显式正交化,最坏情况的时间复杂度为 $O(n^3)$。另外QR等很多求解方法很难并行。Divide & Conquer算法是一种可通过并行方法高效求解三对角矩阵本征值问题的算法,并可提供高精度的计算结果,但该算法的计算时间复杂度为 $O(n^3)$。另一方面,Divide & Conquer算法不能高效地计算部分本征对,但MRRR算法计算 k 个本征对的时间复杂度仅为 $O(nk)$。本文在C1060 GPU上给出了一种基于CUDA的MRRR算法的并行实现,并与LAPACK库中串行MRRR算法的实现进行了比较,取得了20倍的加速比。

2 CUDA执行模型

CUDA是NVIDIA推出的一套并行计算架构,这个架构可以用GPU来解决商业、工业以及科学方面的复杂计算问题,它是一个完整的GPGPU(General GPU)解决方案,提供了硬件的直接访问接口,而不必像传统方式依赖图形API接口来实现GPU访问。CUDA程序开发语言以C语言为基础,并对C语言进行扩展。

CUDA编程模型把CPU作为主机(Host),GPU作为协处理器(Co-processor)或者设备(Device)。一个系统可以有一个或多个设备。相应地,一个CUDA程序分为Host端和Device端两个部分。Host端指在CPU上运行的串行代码,

到稿日期:2011-04-30 返修日期:2011-07-28 本文受国家自然科学基金(60873113),国家高技术研究发展计划(863)(2010AA012301),国家重点基础研究发展规划(973)(2011CB309702)资助。

汪丽杰(1987—),女,硕士,主要研究方向为高性能计算,E-mail:wanglj@sccas.cn;赵永华(1966—),男,博士,研究员,主要研究方向为高性能计算、并行算法与软件实现、并行编程模型。

而 Device 端指在 GPU 上运行的并行代码，在 GPU 上的代码称为内核(Kernel)函数。在这个模型中，CPU 和 GPU 协同工作，CPU 负责逻辑性强的事务处理和串行计算，GPU 专注于高度线程化的并行计算。

每一个 Kernel 由一组大小相同的线程块(thread block)来并行执行，同一个线程块内的线程通过共享存储空间来协作完成计算，线程块之间是相互独立的。运行时，每一个线程块会被分派到一个流多处理器(Stream Multiprocessor，SM)上运行。NVIDIA 的 GPU 中，最基本的处理单元是 SP(Streaming Processor)。为了管理这些线程，SM 利用了一种称为 SIMT(Single Instruction Multiple Thread，单指令多线程)的新架构[5]，图 1 是一个 SIMT 多处理器模型，并且显示了 CUDA 的存储层次。每个 SP 对应一个线程，每个 SM 对应一个或几个线程块，SM 实际执行不以线程块为单位，而以 warp 块为单位，每个 warp 块一般包括 32 个线程。执行时，每发出一条指令，SIMT 单元就会选择一个已经准备好的 warp 块执行，并将下一条指令发送到该 warp 块的活动线程。如果某条指令需要等待，SM 会自动切换到下一个 warp 块来执行，以此来隐藏线程的延迟和等待，以达到大量并行化的目的。

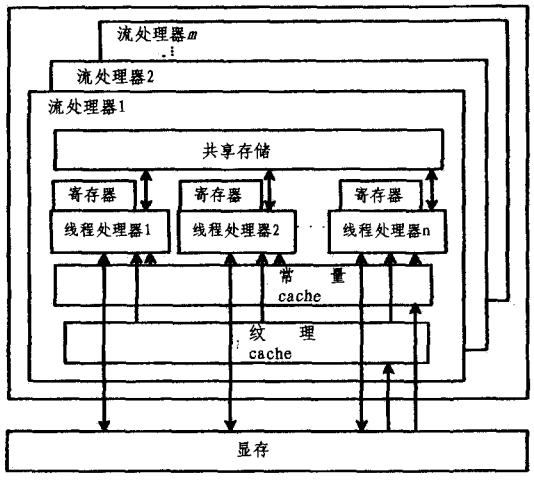

图 1 SIMT 多处理器模型及存储层次结构

3 MRRR 算法

MRRR 算法以本征值为中心，本征值在本征向量之前计算并且按照本征值来展开算法。这一节将给出 RRR 的定义及 MRRR 算法描述(如算法 1 所示)。这两部分都与下面概念有关，首先给出其定义。

定义 1(相对间距 Reldist) 假设 λ_i 和 λ_j 是对称三对角矩阵 $T\in R^{n*n}$ 的两个本征值，且 $\lambda_i<\lambda_j$，称

$$redlist(\lambda_i,\lambda_j)=\frac{|\lambda_i-\lambda_j|}{|\lambda_i|}$$

为 λ_i 和 λ_j 之间的相对间距 Reldist。

定义 2(Relgap) 假设 λ_i 是对称三对角矩阵 $T\in R^{n*n}$ 的本征值，称 λ_i 的 relgap(relative gap)是它所有 Relative Distance 的最小值，即

$$relgap(\lambda_i)=\min\{reldist(\lambda_i,\lambda_j)\,|\,\lambda_i\neq\lambda_j\in\lambda(T)\}$$

算法 1 串行 MRRR 算法实现

输入：一个对称三对角矩阵 T，$T\in R^{n*n}$，相对间距的阈值 tol

输出：T 的本征对 (λ_i,u_i)

1. 计算 T 的 RRR 表示
2. 计算近似本征值 λ_i，并区分 λ_i 是独立的还是成簇的
3. for 每个单独 λ_i do
   计算高精度的本征对
4. for 每个簇 do
   计算新的 RRR 并保存，goto step 2

3.1 RRR

RRR(Relatively Robust Representation)是一个决定本征值本征向量到较高精度的表示，满足表示中小的分量变化仅引起本征对一个相对小的改变。传统的三对角矩阵由它的对角和非对角元素表示，但这不是一个 RRR，对角或非对角元素的一个小的扰动会引起结果的巨大变化。正定三对角矩阵的两对角分解 $T=LDL^t$ 是 RRR，在很多情况下非正定的 LDL^t 也形成 RRR[7]。满足下面两个条件的 LDL^t 分解就是 RRR，由 L 和 D 决定的本征对 (λ,v) 具有较高精度，其中 λ 是本征值，v 是本征向量。L 和 D 发生一个小的相对变化，$l_i\to l_i(1+\eta_i)$，$d_i\to d_i(1+\delta_i)$，$|\eta_i|<\varepsilon$，$|\delta_i|<\varepsilon$，$\varepsilon<<1$，导致 λ 变化 $\Delta\lambda$，v 变化 Δv 满足

$$\frac{|\Delta\lambda|}{|\lambda|}\leqslant k_1 n\varepsilon,\lambda\neq 0, \tag{1}$$

$$|\sin\angle(v,v+\Delta v)|\leqslant\frac{k_2 n\varepsilon}{relgap(\lambda)} \tag{2}$$

式中，k_1、k_2 是常量。RRR 的优点是可以计算较高精度的本征值和本征向量。

3.2 本征值划分及簇的转化

如果 $relgap(\lambda_i)>tol$，则称本征值 λ_i 是独立的，否则称 λ_i 是非独立的，其中 tol 是相对间距的阈值。一组非独立本征值组成了一个簇(cluster)。对于独立的本征值，可以直接计算它所对应的具有较高精度的本征对[2]。对于非独立本征值，由于无法精确地计算其本征对，通过矩阵变换 $L_cD_cL_c^t=LDL^t-\delta I$ 来增大同一簇中的本征值的相对间距，这里 δ 是用来拆开这个簇的值。对其反复进行变换，直到所有本征值都变成独立的。对独立的本征值计算出来的本征向量是正交的，不需要显式正交化。

3.3 表示树

划分本征值及簇的转化过程形成了一个表示树，其中独立的本征值作为叶节点，簇作为内部节点，根节点是初始的输入矩阵和它所需要计算的本征值集合，边对应矩阵变换。

图 2 给出了一个 8 阶对称三对角矩阵求解所有本征对的过程。根节点初始化为需要计算的本征值集合。算法的第一步是将本征值分类。首先 λ_1 到 λ_5 形成一个簇，要计算一个新的 RRR $L_1D_1L_1^t=L_0D_0L_0^t-\tau_1 I$。接下来判断出 λ_6 是一个独立节点，在树中表示为叶节点，对此节点可以直接求其对应本征对到高的相对精度。同样去处理其它簇和叶节点，直到所有本征对都计算出来。串行算法的执行过程类似树的广度优先遍历。

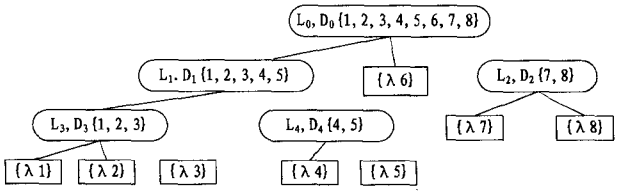

图 2 一个 8 阶实对称矩阵对应的表示树

4 MRRR 的 CUDA 并行算法与实现

本节具体介绍基于 CUDA 的并行 MRRR 算法实现。如上所述，串行算法依次处理表示树中当前层的每个节点，如果是独立节点，计算高精度的本征对；如果是簇，做矩阵变换并且修正这一簇中的本征值。为了适应 CUDA 的并行体系结构，并行方案主要分成两步：第一步拆分簇，直到所有本征值都是独立节点；第二步计算所有本征对。

图 3 是上面 8 阶矩阵对应的两步并行执行，这两部分用横线隔开。对于独立节点，并行方法不立即计算本征对，我们保存它对应的 RRR 以便下一步使用。当所有的本征值都拆分成独立节点，如图 3 所示的 8 阶矩阵拆分成 λ_1 到 λ_8 8 个节点后，并行地求解 8 个节点对应的本征对。对第一步矩阵拆分过程也进行了并行化，下面给出具体并行方法。

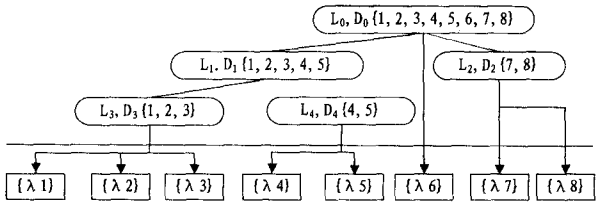

图 3 一个 8 阶实对称矩阵的并行表示树

MRRR 算法的树形结构表明对表示树中同一层的所有节点的处理是独立可并行的，把拆分簇的过程分成两个部分，首先找到表示树中这一层的所有簇，然后并行地对这些簇进行矩阵变换求解新的 RRR 及修正簇的中本征值。

簇的转化先做矩阵变换求出新的 RRR，然后修正簇中的元素，分别由 qds 转换和二分法来实现。求解较高精度的本征对也是基于 qds 转换实现的。因此需要实现 qds 转换以及二分法的并行。qds 转换的内部存在一个有依赖的循环，每次都需要上一次迭代的运行结果，因此内部不能实现并行。但是表示树中每一层有多个簇，可以同时对这些簇进行 qds 转换。二分法各个迭代之间没有相互依赖是一种可高效并行的方法，可以让一个本征值对应一个 thread 来并行处理。

用 pqds 表示在 GPU 上执行的并行 qds 函数，用它来并行计算表示树当前层所有新簇的 RRR，用 pbis 表示并行二分算法，pbis 函数用来并行修正表示树当前层所有簇中的本征值，pvec 表示求解本征对的 Kernel 函数，在我们设计的并行 MRRR 算法中，用 pvec 并行求解所有本征对。

算法 2 MRRR 算法在 GPU 上的并行实现方法

输入：一个对称三对角矩阵 $T, T\in R^{n*n}$，相对间距的阈值 *tol*

输出：T 的本征对 (λ_i, u_i)

[host]

1. 计算 T 的 LDL^t 表示，初始化表示树的根节点为所有要求解的本征值集合；
2. 按照 *relgap* 划分这一层的所有簇为新的簇或独立节点，并保存新簇的首末元素位置；
3. if(没有新的簇形成)
   goto step 6

[device]

4. (pqds)并行的计算并保存所有新簇的 RRR；
5. (pbis)并行修正所有新簇的本征值；
   goto step 2
6. (pvec)并行求解高精度的本征值本征向量。

主机任务：如算法 2 中描述，先计算 T 的 LDL^t 表示，它是一个串行过程，放在 CPU 上执行(step1)。接下来把表示树这一层的所有簇进行分类(step2)，由于分类过程有顺序性，且用时少，把这一部分也放在 CPU 上。分成独立节点或新簇，用数组保存所有新簇的起止元素位置，这个数组将会传给 GPU，使得只有对应本征值在新簇中的线程才做矩阵转换及修正本征值。

主机的另一个任务是进行判断(step3)，如果没有形成新簇，即所有本征值都已分成了独立节点，则调用求解本征对的 pvec 函数，否则调用 pqds 函数。

设备任务：根据主机传过来的保存所有新簇起止元素位置的数组来对簇进行拆分，每个线程对应一个簇进行 qds 计算并保存所有新簇的 RRR(step4)，然后让每个线程对应簇中的一个本征值用二分法修正这些本征值(step5)。再回到主机去划分新簇，直到没有新簇形成，则调用 pvec 函数来并行求解高精度的本征值本征向量(step6)。

在数据存储方面，由于 shared memory 很小、矩阵规模较大，在实际中较少使用，这也是以后需要改进的地方。对 global memory 使用方面做了一些优化。在 CUDA 的存储器结构中，全局存储器是最慢的，但是其容量是最大的，因此全局存储器的合理使用对 CUDA 程序性能至关重要。并行算法采用了 CUDA 的合并访问机制来加速全局存储访问。在求解本征对过程中，要给每个线程分配一个工作空间，传统方式是给每个线程分配连续的工作空间，当读取第 i 个工作空间时，各个线程读取的地址是不连续的，造成了很大的等待时间。我们的工作空间分配方法是每个线程有分散的工作空间，但所有线程对应的工作空间的同一个项是连续的，这样使得相邻的线程访问相邻的工作空间，减少了数据访问时间。

5 实验结果

基于 CUDA 实现了 MRRR 算法的并行，并且和 LAPACK 库中的 sstemr 进行了比较。测试系统是 1 个有 90 个计算节点的 GPU cluster，每个节点有 2 个 Xeon 5410 四核 CPU 和 2 块 Tesla C1060 GPU 卡。测试了两种类型的矩阵，一个是 Wilkinson 矩阵，另外一个是对角线元素是 4、副对角线元素是 1 的矩阵。7 阶 Wilkinson 矩阵的对角线元素是 3、2、1、0、1、2、3，副对角线元素是 1，Wilkinson 矩阵有很多非常接近的本征值，有非常显著的本征值聚集的特点，因此用这个矩阵来检查并行代码的性能。

表 1 Wilkinson 矩阵串行和并行运行时间(ms)

比较值	对应不同规模的 Wilkinson 矩阵运行时间对比				
	512	1024	2048	4096	7168
sstemr	4.59e+2	2.026e+3	9.015e+3	1.485e+4	3.194e+4
cuda_mr	1.21e+2	4.531e+2	1.447e+3	2.190e+3	4.249e+3
speedup	3.8	4.47	6.23	6.78	7.52

用 sstemr 表示串行 MRRR 算法，cuda_mr 表示基于 GPU 的并行算法。表 1 为 Wilkinson 矩阵对应的不同规模的矩阵串行运行时间、并行的运行时间及加速比，运行时间单位是 ms，图 4 是对应的运行时间对比图。如表 1 所列，对于 Wilkinson 矩阵，并行 MRRR 算法取得了 7.5 倍加速。表 2 和图 5 是对角元素是 4 的矩阵的运行时间对比，运行时间单位是 ms，如图 2 所示，对于这个矩阵，并行方法取得了超过 20 倍的加速。Wilkinson 矩阵的本征值聚集度大，分成的簇很多，而对角线是 4 的矩阵本征值聚集度不大，分成的簇较少，当前的算法对于聚集度不大的矩阵效果更好。原因在于每一

簇进行矩阵转换及求解本征向量的过程中都要用到对应的 D_i, L_i，对于聚集度不大的矩阵，拆分过程很快并且很多节点对应一个 D_i, L_i；相反，聚集度大的矩阵，拆分成独立节点的时间较长，分成的簇很多，对应同一个 D_i, L_i 的节点也较少，各个线程进行到同一步时，读取相同的 D_i, L_i 数据会比读取不同的节省很多数据读取时间。在计算精度方面，计算出来的结果与串行 sstemr 的结果基本相同，误差不超过万分之一。

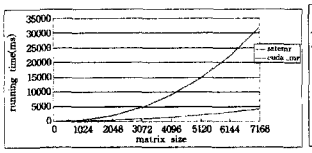

图 4 Wilkinson 矩阵运行时间对比

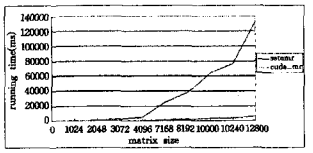

图 5 对角线元素是 4 的矩阵运行时间对比

表 2 对角线元素为 4 的矩阵串行和并行运行时间(ms)

比较值	对应不同规模的对角元素是 4 的矩阵运行时间对比					
	1024	2048	4096	8192	10240	12800
sstemr	1.40e+2	7.63e+2	5.01e+3	3.78e+4	7.69e+4	1.33e+5
cuda_mr	4.06e+2	1.39e+2	3.21e+2	2.13e+3	3.80e+3	6.38e+3
speedup	3.44	5.5	9.6	17.7	20.23	20.76

结束语 本文给出了一种基于 CUDA 的并行 MRRR 算法，用来求解对称三对角矩阵本征值问题，取得了较好的加速效果。实验证明，MRRR 算法适合在 GPU 上并行。但此方法在数据存储及并行策略上还有待改进，希望日后能够在保证精度的基础上改进方法以取得更大加速。

参 考 文 献

[1] Cuppen J J M. A Divide and Conquer Method for the Symmetric Eigenproblem[J]. Number, 1981, 36: 177-195

[2] Dhillion I S. A new $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem[D]. EECS Department University of California, Berkeley, 1997

[3] Dhillon I S. The Design and Implementation of the MRRR Algorithm[J]. ACM Transactions on Mathematical Software, 2006, 32(4)

[4] Dhillon I S, Parlett B N. Multiple Representations to Compute Orthogonal Eigenvectors of Symmetric Tridiagonal Matrices [J]. Linear Algebra and its Applications, 2006, 378(1): 1-28

[5] NVIDIA Corporation. NVIDIA CUDA Programming Guide 2.0 Final[M]. NVIDIA Corporation, 2008

[6] Gu M, Eisentat S C. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem[J]. SIAM J. Mat. Anal. Appl, 1995, 16(1): 172-191

[7] Dhillon I S, Parlett B N. Orthogonal Eigenvectors and Relative Gaps[J]. SIAM J. Matrix Anal. Appl, 2003: 858-899

(上接第 278 页)

重构平台上的执行性能。Gaussian Elimination 测试程序中具有 3 层循环迭代，计算量相对较大，当数据规模增大时，硬件执行性能优势逐渐明显，当数据规模达到一定的量时，可并行任务划分在可重构器件上执行比在 CPU 上执行能够获得更高的性能。

表 5 Gaussian Elimination 测试结果

测试程序	硬件执行时间		软件执行时间		划分结果	
	估算	实际	估算	实际	估算	实际
256 * 256	0.153	0.169	0.131	0.145	S	S
512 * 512	0.458	0.499	0.433	0.472	S	S
1024 * 1024	1.680	1.815	1.714	1.849	H	H
2048 * 2048	6.622	7.091	7.048	7.498	H	H

S 表示在 CPU 上执行，H 表示在可重构器件上执行

从上述 8 种测试程序的实验结果中可以看出，本文提出的基于程序性能评估的软硬件任务划分方法能够有效指导应用程序的任务划分，实现提升应用程序执行性能的目标。

结束语 可重构系统的高性能和灵活性为应用程序提供了良好的计算平台，软硬件任务划分是有效利用可重构资源的关键技术之一。本文提出的采用任务流图的方式对应用程序结构进行描述，能够屏蔽可重构计算系统中不同计算部件的不同设计语言和设计方法给任务划分带来的困难。通过对 FPGA 计算开销、配置开销、通信开销的预评估测试，能够减少划分方案的搜索范围，提高任务划分方案的准确性。最后通过改进的模拟退火算法能够形成任务与执行期间之间的有效映射。实验结果表明，该划分方法具有较好的划分效果，以及较好的算法收敛速度。

参 考 文 献

[1] Hartenstein R. A decade of reconfigurable computing: a visionary retrospective[A]// Proceedings of the conference on Design Automation & Test in Europe, 2001[C]. New York, USA: ACM Press, 2001: 642-649

[2] Compton K, Hauck S. Reconfigurable computing: a survey of systems and software[J]. ACM Computing Surveys, 2002, 34 (2): 171-210

[3] 熊志辉，李思昆，陈吉华. 遗传算法与蚂蚁算法动态融合的软硬件划分[J]. 软件学报，2005，16(4)：503-512

[4] 马平. 可重构系统中的任务划分和任务调度的研究[D]. 天津：河北工业大学，2006

[5] Li Y, Callahan T, Darnell E, et al. Hardware-Software co-design of embedded reconfigurable architecture[A]// Proceedings of Design Automation Conference, 2000[C]. Los Angeles, California: ACM, 2000: 507-512

[6] 沈英哲. 可重构计算系统中软硬件代码划分技术研究[D]. 合肥：中国科技大学，2007

[7] 袁爱平，傅明. 嵌入式系统软硬件划分方法探索[J]. 计算机应用，2008，28(9)：2427-2429

[8] Girkar M, Polychronopoulos C D. The hierarchical task graph as a universal intermediate representation[J]. International Journal of Parallel Programming, 1994, 22(5): 519-551

[9] Zhang Dan, Zhao Rong-cai, Han Lin, et al. A Parallelization Cost Model for FPGA[J]. Advanced Materials Research, 2011, 181-182, part 2: 623-628

[10] Ingber L. Very fast simulated annealing [J]. Math Compute Modeling, 1989, 12(8): 967-973

[11] Noori H, Mehdipou F, Murakam K. A Reconfigurable Functional Unit for an Adaptive Dynamic Extensible Processor[A]// Proceedings of 16th IEEE International Conference on Field Programmable Logic and Applications 2006[C]. Madrid, SPAIN: IEEE Press, 2006: 781-784

[12] Livermore Benchmarks[EB/OL]. http://www.netlib.org/benchmark/livermorec, 1992-10-20