基于GPU的分子动力学模拟并行化及实现

费 辉[1,2,3] 张云泉[1,2] 王 可[1,2] 许亚武[4]
(中国科学院软件研究所并行软件与计算科学实验室 北京 100190)[1]
(中国科学院软件研究所计算机科学国家重点实验室 北京 100190)[2]
(中国科学院研究生院 北京 100190)[3] (广州大学网络与现代教育技术中心 广州 510006)[4]

摘 要 分子动力学模拟作为获得液体、固体性质的重要计算手段,广泛应用于化学、物理、生物、医药、材料等众多领域。模拟体系的复杂性和精确性的需求,使得计算量巨大,耗费时间长。并行计算是加速大规模分子动力学模拟的重要途径。GPU以几百GFlops甚至上TFlops的运算能力,为分子动力学模拟等的计算密集型应用提供了新的加速方案。提出了一种基于GPU的分子动力学模拟并行算法——oApT-AD,并在OpenCL和CUDA框架下加以实现。性能测试显示,在Tesla C1060显卡上,该算法在OpenCL框架下的实现相对于CPU的串行实现,最高达到120倍加速比。通过对比发现,该算法在CUDA上的性能与OpenCL基本相当。同时,该算法还可以扩展到两块及以上的GPU上,具有良好的可扩展性。

关键词 分子动力学,GPU,OpenCL,CUDA,原子分解法
中图法分类号 TP312 **文献标识码** A

Parallel Algorithm and Implementation for Molecular Dynamics Simulation Based on GPU

FEI Hui[1,2,3] ZHANG Yun-quan[1,2] WANG Ke[1,2] XU Ya-wu[4]
(Lab. of Parallel Software and Computational Science, ISCAS, Beijing 100190, China)[1]
(State Key Lab. of Computer Science, ISCAS, Beijing 100190, China)[2]
(Graduate University of Chinese Academy of Science, Beijing 100190, China)[3]
(Network and Modern Education Technology Center, Guangzhou University, Guangzhou 510006, China)[4]

Abstract Molecular Dynamics Simulation is an important method for acquiring liquid and solid atoms' properties. This method has been widely used in the fields of chemistry, physics, biology, medicine and materials. The complexity and accuracy demand causes enormous workloads. Parallel computing is a feasible way to speedup large-scale molecular dynamics simulation. With hundreds of GFlops or even TFlops performance, GPU can speed up computing-intensive applications. This paper presented a parallel algorithm named oApT-AD, and we implemented it on GPU under OpenCL and CUDA Framework. The experiment results show that the oApT-AD algorithm can achieve 120 speedup on GPU Tesla C1060 under OpenCL Framework, compared to that on CPU. And we also implemented the oApT-AD algorithm on GPU under CUDA Framework. The implement under OpenCL Framework provides almost the same performance as the implement under CUDA Framework. Moreover, our algorithm can be extended to two or more GPUs, with good scalability.

Keywords Molecular dynamics, GPU, OpenCL, CUDA, Atom decomposition

1 引言

分子动力学模拟,是指对于原子核和电子所构成的多体系统,用计算机模拟原子核的运动过程,从而得到计算系统的结构和特性,其中每一原子核被视为在其它所有原子核和电子所提供的经验势场作用下按牛顿定律运动[1]。分子动力学模拟作为获得液体、固体分子性质的常用计算手段,广泛应用于化学、物理、医药、材料、能源、生物等众多领域当中。分子动力学模拟能够跟踪分子的运动状态,并能够通过统计地分析速度、温度等性质,阐述理论中的难点。但是有限的计算能力一直是制约这方面研究发展的瓶颈[2]。分子动力学模拟的主要特点是计算量庞大,产生这种状况的原因有以下两个方面[3]:一是需要在尺寸很小的分子尺度上进行分子模拟,模拟的体系日趋复杂;二是为了能够精确地描述分子的运动状态,

到稿日期:2010-10-02 返修日期:2010-12-25 本文受国家863计划项目(2006AA01A125,2009AA01A129,2009AA01A134),国家重大专项核高基项目(2009ZX01036-001-002)资助。

费 辉(1985—),男,硕士生,主要研究方向为并行算法与并行软件,E-mail:feihui.ustc@gmail.com;张云泉(1973—),研究员,博士生导师,主要研究方向为高性能计算及并行数值软件、并行计算模型、并行数据库、海量数据并行处理;王 可(1981—),博士,副研究员,主要研究方向为并行计算;许亚武(1967—),男,高级工程师,主要研究方向为计算机网络。

要求模拟的时间步要足够的细，这就需要运行较多的时间步长才能达到良好的模拟效果。有相关研究表明，进行分子动力学模拟时，约90%以上的时间花费在计算分子间的作用力上[4]。

分子动力学模拟的并行算法主要是针对力的计算进行划分，有原子分解法、力分解法和空间分解法3种[5]。这3种算法的主要区别在于它们对并行任务的划分有所不同：原子分解法是将所有原子编号，进行平均划分；力分解法是将计算原子受力的矩阵以一定的技巧按块分割划分；区域分解法是按物理区域进行任务划分。

本文提出了一种基于原子分解法的GPU并行算法——oApT-AD，通过对齐的数据的存储和访问以及对GPU共享存储器的有效利用，该算法能够很好解决数据冲突、访存缓慢等问题。实验结果表明，该算法获得了很好的加速效果，并且具有良好的可扩展性。本文第2节介绍分子动力学模拟并行方案的相关工作；第3节介绍我们提出的分子动力学模拟并行算法——oApT-AD的设计与实现；第4节介绍实验结果，并对结果进行分析；最后对全文进行总结。

2 相关工作

2.1 分子动力学模拟常用算法

在分子动力学模拟中，系统中分子的速度、位移是通过对牛顿运动方程积分得到的，通过解牛顿第二定律的微分方程，可以得到原子的具体运动细节。通常采用有限差分法来求解运动方程。常用的算法有以下几种[1]：Verlet算法，Velocity-Verlet算法，Leap-frog算法，Gear算法，Beeman算法，Rahman算法。

本文采用的算法为Leap-frog算法，Leap-frog算法是Verlet算法的一种变化，与Verlet算法相比，计算量稍小，算法更新位置的误差为$O(\delta t^4)$，且比较容易实现，在分子动力学模拟中被广泛使用。速度和位移的求解如式(1)和式(2)所示。

$$r(t+\delta t)=r(t)+v(t+\frac{1}{2}\delta t)\delta t \tag{1}$$

$$v(t+\frac{1}{2}\delta t)=v(t-\frac{1}{2}\delta t)+a(t)\delta t \tag{2}$$

在分子动力学模拟中，分子间力的作用常用伦纳德-琼斯势(L-J势)[6]表示，如式(3)所示。

$$U(r_{ij})=4\varepsilon[a(\frac{\sigma}{r_{ij}})^{12}-b(\frac{\sigma}{r_{ij}})^{6}] \tag{3}$$

式中，ε为作用势能；σ为作用特征长度(σ一般为分子直径)；r_{ij}为分子i到j的距离；系数a、b表示斥力项和引力项的相对大小，一般情况$a=b=1$。由此，分子i受到分子j的作用力F_{ij}如式(4)所示。

$$F_{ij}=-\nabla U(r_{ij}) \tag{4}$$

分子的受力计算完成后，就可以更新分子的速度和位置信息，对分子的运动方程进行积分。分子动力学模拟中，分子运动遵循牛顿运动定律，如式(5)所示。

$$m_i\frac{dv_i}{dt}=\sum_j F_{ij}+F_{external} \tag{5}$$

2.2 分子动力学模拟并行化研究进展

1999年，IBM为生物分子学模拟启动了Blue Gene[7]项目，制造了大型机系统来满足计算的需求。2005年，BG/L系统[8]设计完成，该系统由65536个节点组成，节点嵌入IBM PowerPC 440处理器核，系统的峰值计算达到360 teraflops。文献[9]实现了分子动力学模拟软件NAMD，用于在大规模并行计算机上快速模拟大分子体系，其通过Charm++实现了动态负载平衡，能够很容易地扩展到数百乃至数千个处理器上运行。NAMD还配有分析辅助软件VMD[10]。

文献[11]利用GPU的图形渲染方法实现了分子动力学模拟，达到了较好的加速效果，但是具有局限性。文献[12]采用原子分解法，实现了分子动力学模拟在CUDA架构下的加速。文献[2]采用MPI+GPU-MD的并行方案，基于空间分解法在集群系统中实现了分子动力学模拟的加速。已有的这些利用GPU算法的共同点，就是只需要一次从主机到设备的输入数据拷贝。但是这些算法都没有基于较新的硬件平台，没有充分利用GPU合并访问模式和共享存储器的特性，致使在GPU内部会出现访存冲突、访存带宽低效的问题，导致性能下降。

3 oApT-AD算法及实现

3.1 分子动力学模拟串行算法

分子动力学模拟在CPU上实现的一般过程如算法1所示。

算法1(CPU串行算法)

1) 读取分子状态信息，设置模拟参数；
2) 计算分子的初始状态信息；
3) For all time steps do；
4) 使用式(1)更新所有分子的位移信息；
5) 使用式(3)和式(4)计算分子的受力情况，并使用式(2)和式(5)更新分子的速度信息；
6) End for

算法1来自文献[6]，描述了在CPU上实现分子动力学模拟的过程。值得注意的是，在上述算法中，由于原子之间的作用是相互的，因此有$F_{ij}=-F_{ji}$，其中F_{ij}表示原子j对原子i的作用力，F_{ji}表示原子i对原子j的作用力。于是计算力矩阵相当于一个上三角或下三角的矩阵，即如果计算了F_{ij}，F_{ji}就不需要重新计算了，力的计算时间可以减小一半。

3.2 oApT-AD算法

基于原子分解法在GPU上实现并行化是一个很好的选择。原子分解法适合于GPU应用，表现在3个方面：1)区域分解法在分子移除网格时，处理器之间需要交换原子信息，适合于cluster，不适合GPU进行加速；而对于力分解法，处理器之间依赖关系比较强，需要处理读写冲突问题，同样不适合GPU加速。2)利用原子分解法可以达到较好的负载平衡和可扩展性。3)GPU支持共享存储结构，原子分解法在计算时适用于共享存储的结构，且实现时不会产生读写冲突。

在GPU的编程模型中，线程的动态调度和执行可以交给GPU来完成，根据这个特点，我们用一个线程处理一个分子，提出oApT-AD(one Atom per Thread based Atom Decomposition)并行算法，该算法的具体描述如算法2所示。

算法2(oApT-AD并行算法)

1) 读取分子状态信息，设置模拟参数；
2) 计算分子的初始状态信息；
3) 从CPU内存拷贝分子信息到GPU显存；

4) For all time steps do;

5) 执行 GPU 上的 MoveKernel 算法;

6) 执行 GPU 上的 ForceKernel 算法;

7) 从 GPU 显存读取分子信息到 CPU 内存;

8) End for

算法 2 运行在 CPU 中,调用了 MoveKernel 和 ForceKernel 算法;算法 MoveKernel 和 ForceKernel 运行在 GPU 中。CPU 端的任务有两个,一是初始化所有原子的信息,然后配置 GPU 环境,即设置必要的模拟参数,最后将所有原子的速度、位移信息拷贝到 GPU 显存中。二是当 GPU 运算结束后,从 GPU 显存读出计算结果数据放到 CPU 内存中。GPU 端的任务是,每个线程计算分配的原子的速度与位移,将计算结果保存,其中 MoveKernel 算法计算分子的更新位移,ForceKer-nel算法计算分子的更新速度。该算法很好地刻画了分子动力学模拟中计算的详细过程,每一个时间步中,由 GPU 进行计算,然后将计算结果交给 CPU 进行处理。在这个过程当中,由于各个分子信息是独立的,每个线程在计算时也就相对独立。

算法 3(MoveKernel 算法)

1) get_global_id,得到线程 id 为 x;

2) 访问编号为 x 的分子的速度信息;

3) 利用式(1)更新分子 x 的位移信息;

在算法 3 中,所有分子信息的存储方式采用 64Byte 对齐,在访问数据时充分利用 GPU 的合并访问模式,如图 1 所示。如果线程访问地址是连续的,并且是 64Byte 段对齐的,合并访问模式会导致一次 64Byte 合并传输,这样大大地提高了可利用的带宽,能够有效地减少 MoveKernel 算法在 GPU 中的计算时间。

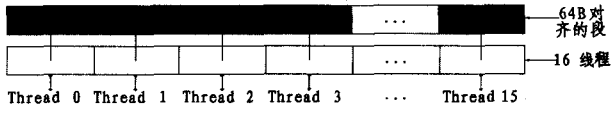

图 1 GPU 的合并访问模式

算法 4(ForceKernel 算法)

1) get_global_id,得到线程 id 为 x;

2) 设置共享存储器大小 SharedSize 等于 BlockSize;

3) For i from 0 to n−1;

4) 读取从编号 i 到编号 i+SharedSize−1 的分子信息到共享存储器;

5) 使用式(3)和式(4)计算共享存储器内的 SharedSize 个分子 y 对 x 的作用力 F_{xy};

6) $F_x=F_x+F_{xy}$,i=i+SharedSize;

7) End for

8) 使用 F_x,由式(2)和式(5)更新分子 x 的速度信息;

在算法 4 中,充分利用了 GPU 中的共享存储器,经测试,当共享存储空间 SharedSize 与线程块 BlockSize 大小相等时,性能最佳。当使用了共享存储器之后,全局存储器的分子信息会分批次地放到共享存储器中,如图 2 所示。访问共享存储器的速度比全局存储器快很多,这样,线程在计算分子受力时不用从全局读取分子信息,而直接在共享存储器上读取,在访存效率上有了很大提高,使得 ForceKernel 算法在 GPU 中的执行时间有所减少。

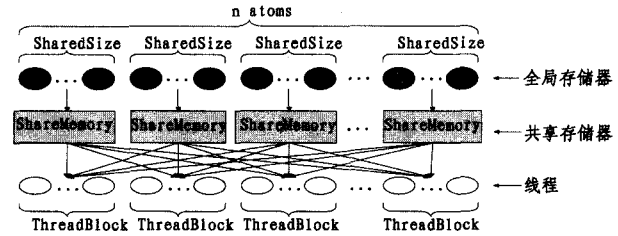

图 2 使用共享存储器计算分子之间的作用力

4 实验与分析

实验的编译及运行环境如表 1 所列。

表 1 实验编译及运行环境

操作系统	Ubuntu 4.3.3-5
编译器	gcc4.3.3
	icc11.1
	nvcc3.0
CPU	Intel(R) Xeon(R) CPU X5472 @ 3.00GHz
GPU	Tesla C1060
	Quadro FX 4800

在编译 OpenCL 实现部分时选用的编译器是 gcc;编译 CPU 实现部分时选用的编译器是 icc,并用-O3 和-axSSE4.1 选项做了优化;编译 cuda 程序时选用的编译器是 nvcc。Tesla C1060 的单精度浮点性能是 933 GFlops;Quadro FX 4800 的单精度浮点性能是 462GFlops。

分子动力学模拟中设置的参数为 $density=0.4$,$rcut=5.0$,$dt=0.001$,参数的单位都是无量纲的约简单位。

4.1 oApT-AD 算法在 OpenCL 上的分析

表 2 为 OpenCL 框架下利用 CPU+GPU 模拟 1 个时间步长的运行时间分布。在表 2 中,Particles 列表示模拟的分子个数,EnvTime 列表示 GPU 运行环境的创建与销毁时间,CpyTime 列表示主机内存与 GPU 显存之间数据传输的时间,CompTime 列表示 GPU 的计算时间,TotalTime 列表示利用 CPU+GPU 计算的总时间。从表 2 中可以看出,在分子数较少时,程序的运行时间花费在 GPU 环境的创建与销毁上所占的比例比较大,这样不利于加速。因此在利用 GPU 进行计算时不宜频繁创建、销毁 GPU 运行环境。在分子数较多,并且模拟时间步较多的情况下,GPU 环境的创建与销毁的时间会掩盖得越好,得到的相对性能就越高。

表 2 模拟 1 个时间步长,利用 GPU 进行加速的时间分布

Particles	CPU+GPU(ms)			
	EnvTime	CpyTime	CompTime	TotalTime
2048	45.45	0.46	2.34	48.25
4096	44.92	0.48	3.99	49.39
8192	46.06	0.57	11.88	58.51
16384	47.73	0.73	34.82	83.28
32768	48.14	0.84	109.01	157.99
65536	49.2	1.33	388.2	438.73
131072	51.31	2.24	1530.53	1584.08

加速比的计算方法如式(6)所示。

$$Speedup=\frac{T_{CPU}}{T_{CPU+GPU}} \tag{6}$$

图 3 描述了 OpenCL 框架下,不同分子规模随 SharedSize 大小变化的加速比情况。从图中可以看出,当分子规模较小时,宜选择较小的 SharedSize,例如,分子规模为 2048 和 4096,SharedSize 选择为 32 时,加速效果最好。随着分子规

模增大，SharedSize 也应该随着增大，例如，当分子规模为 8192，SharedSize 选择为 64 时，加速效果最好。当分子规模达到 16384 和 32768，SharedSize 选择为 128 时，加速效果最好。当分子规模达到 65536 及以上，SharedSize 选择为 256 时，加速效果最好。

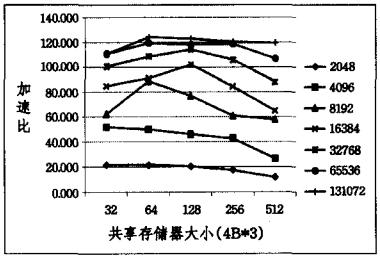

图 3　不同分子规模随 SharedSize 变化的性能对比

表 3 记录了在 SharedSize 为 256 的情况下，模拟 100 个时间步时 CPU 和 CPU+GPU 运行的时间分布，CPU 列表示 CPU 串行模拟的时间，CPU+GPU 列表示 CPU+GPU 的模拟时间。从表中可以看出，当分子数比较小候，CPU+GPU 总的执行时间中 GPU 计算时间所占比重比较小，导致加速比较低，当分子数增多时，GPU 计算时间所占比重逐渐增大，加速比也随着增大。当 GPU 计算时间几乎为 CPU+GPU 执行时间时，加速比趋于不变，即达到了加速极限。分子数再增多时 CPU 的执行时间和 CPU+GPU 的执行时间将以同样的倍数增长，加速比基本保持不变，最高可达 120 倍加速比。

表 3　模拟 100 个时间步长，利用 GPU 加速结果

particles	CPU(s)	CPU+GPU(s)	speedup
2048	5.33	0.30	17.65
4096	20.07	0.47	42.82
8192	76.52	1.26	60.82
16384	296.99	3.53	84.15
32768	1166.43	11.02	105.85
65536	4621.06	38.98	118.54
131072	18382.37	153.01	120.14

4.2　oApT-AD 算法在 OpenCL 和 CUDA 上的对比分析

图 4 描述了 oApT-AD 算法在 CUDA 和 OpenCL 架构下的加速比趋势对比，GPU 为 Tesla C1060，SharedSize 为 256。从图中可以看出，oApT-AD 在 CUDA 架构和 OpenCL 架构下的加速效果基本相当。当分子数较少时，oApT-Ad 算法在 OpenCL 上的性能略低于 CUDA；当分子数较多时，该算法在 OpenCL 上的性能略高于 CUDA。因此，对于分子动力学模拟来说，GPU 应用可以在 OpenCL 框架下实现，这样的实现在性能上与 CUDA 框架下的实现基本相当，但是具有更好的可移植性。

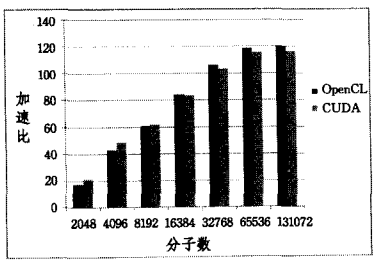

图 4　OpenCL 架构与 CUDA 架构的加速对比图

4.3　oApT-AD 算法的扩展性分析

oApT-AD 算法还有一个特性，即扩展性较强。若有两个以上的 GPU 参与计算，则可以按照 GPU 的单浮点计算能力进行计算任务划分。我们将该算法扩展到 2 个 GPU 上进行了实现，2 块 GPU 分别为 Tesla C1060 和 Quadro FX 4800，性能统计的结果如图 5 所示，SharedSize 大小设置为 256。在分子数较少时，由于 2 个 GPU 所需环境的创建与销毁时间较长，因此加速比效果不好，低于 1 个 GPU 的加速效果。当分子数较多时，GPU 计算的时间所占比重较大，2 个 GPU 的加速效果就显现出来了。由于 Quadro FX 4800 的性能较 Tesla C1060 差，根据 2 块 GPU 的峰值性能：Tesla C1060 的单精度浮点性能峰值 933 GFlops，Quadro FX 4800 的单精度浮点性能峰值 462GFlops，由式(7)可以算出，利用 Tesla C1060+Quadro FX 4800 达到的性能理论上应该为 TeslaC1060 的 1.5 倍。

$$P=\frac{Peak_{tesla}+Peak_{\text{Quadro FX}}}{Peak_{tesla}} \tag{7}$$

在实际的模拟运算中，在分子数为 65536 和 131072 时，Tesla C1060+Quadro FX 4800 的性能几乎为 Tesla C1060 的 1.5 倍，达到了最大性能。由于 C1060+Quadro FX 4800 这 2 块 GPU 相对于 TeslaC1060 的 1 块 GPU 来说，多出了 Quadro FX 4800 的数据传输时间和运行环境的创建与销毁时间，因此在分子数较少时实际性能低于理论值 1.5，而当分子数较多时，这部分时间可以忽略，几乎达到了理论值 1.5。

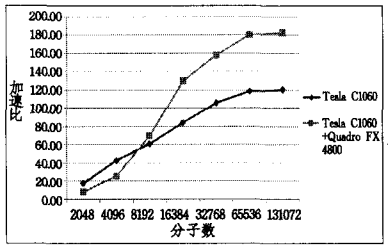

图 5　OpenCL 架构下 oApT-AD 算法扩展性示意图

结束语　本文提出了一种基于原子分解法的分子动力学模拟并行算法 oApT-AD，并在 OpenCL 和 CUDA 框架下进行了具体的实现。性能测试结果显示，分子动力学模拟在跨平台的 OpenCL 框架下利用 GPU 进行实现，可以得到良好的加速效果，最高可达到 120 倍加速比。同时，将 OpenCL 框架下的实现与 CUDA 架构下的实现进行了比较，测试结果显示，OpenCL 架构下的实现与 CUDA 架构下的实现性能基本相当。该算法的可扩展性也较好，可以扩展到多个 GPU 上，在实验中扩展到了 2 块 GPU 上，当分子数较多、计算量较大时，可以获得最理想的性能。对于分子动力学模拟来说，考虑到程序代码的可移植性，选择在 OpenCL 框架下实现会更好。

参 考 文 献

[1]　刘玉华，朱如曾，周富信，等. 分子动力学模拟的主要技术[J]. 力学进展，2003，33(1)：65-73

[2]　陈飞国，葛蔚，李静海. 复杂多相流动分子动力学模拟在 GPU 上的实现[J]. 中国科学 B 辑：化学，2008，38(12)：1120-1128

[3]　王小伟，郭力，杨章远. 近程作用分子动力学模拟的两级并行[J]. 计算机与应用化学，2003，20(5)：639-642

[4]　Wu Jiang-tao，Liu Zhi-gang，Zhao Xiao-ming. Computation efficiency of some short-range force algorithms in molecular dynamic simulation[J]. Journal of Xi'an Jiaotong University，2002，36(5)：477-481

(下转第 287 页)

拟人脑视神经区域，V1，V2，V4 产生与位置和尺度无关的不变量，确认抽象的目标，每一步骤都设有过滤层、非线性层和特性池层，每个网络徊路会由一到三步骤和跟随的分类器组成。

在 Virtex6 FPGA 的硬件上，以 7W 功耗模拟 500 万个神经元，得到 30 fps 的速度，轻易地把 100 万像素的二维图像实时地识别并转换成三维目标对象，计算结果送入手提电脑，用在自动驾驶导航，达到了高性能计算的效果。

我们也在研发一种实时数据采集系统(见图 4)，使用嵌入式计算机，通过 CAN 网络和互联网，管理分散在多个地点的数百台机器。大量的实时数据在收集的过程中逐步转换成简洁的管理资讯。可惜资源有限，投入的人力物力以及个人的精力都不够，除了部分通信软件外，暂时未有可以报告的成绩。

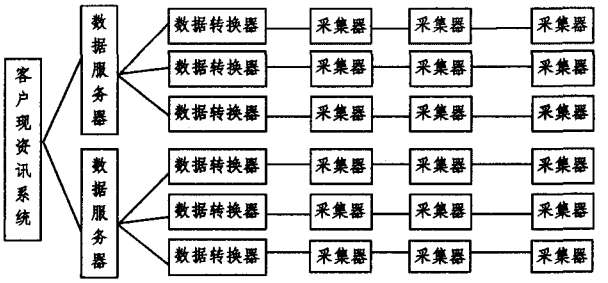

图 4　实时数据采集系统

仿生电脑的范围很庞大，可具体研究的项目很多，如果有其他单位或学校有兴趣，我们很乐意与大家分享经验和无偿地共用取得的国内外知识产权。

结束语　图灵-冯诺曼模型用电子机械替代人类的思维，创造了智能机器，打开了资讯时代的大门。

在对算法抽象的时候，冯诺曼体系把数据和程序分离，引入冗余的地址参数，降低了变换效率。

神经网络以联结表达数据间的关系，牺牲系统的通用性，换回变换速度。

仿生电脑就要在这两个极端之间取得平衡。

参 考 文 献

[1] 漆锋滨. E 级高性能计算机面临的挑战[R]. 全国高性能计算大会报告，2009

[2] Turing A. MComputing machinery and intelligence[J]. Mind, 1950,59:433-460. http://www.loebner.net/Prizef/TuringArticle.html,

[3] Shannon C E. A mathematical theory of communication[J]. Bell System Technical Journal,1948,27:379-423,623-656

[4] 杨学军. 存储墙问题的思考[R]. HPCC 2009 年全国高性能计算大会报告，2009

[5] Peter N. Glaskowsky NVIDIA's_Fermi:The_First_Complete_GPU_Architecture [OL] . http://www.nvidia.com/content/PDF/fermi_white_papers/

[6] 蒋宗礼. 人工神经网络导论 [M]. 北京：高等教育出版社，2003

[7] Fabio Benfenati Synaptic plasticity and the neurobiology of learning and memory [J]. ACTA BIOMED,2007,78(Suppl 1):58-66

[8] Brent Przybus FPGA Families[OL]. http://china.xilinx.com/support/documentation/

[9] 李奕权. 仿生电脑的神经脊[P]. CN101339627

[10] 柯熙正，席晓莉. 无线激光通信概论[M]. 北京：北京邮电大学出版社，2004

[11] Piazza G,et al. Design of a monolithic piezoelectrically actuated. microelectromechanical tunable. vertical-cavity surface-emitting laser[J]. OPTICS LETTERS,2005,30(8)

[12] Huang M C Y, Zhou Ye , Chang-Hasnain C J. Nano electro-echanical optoelectronic tunable[J]. VCSEL. OPTICS EXPRESS, 2007, 1222. http://www.eecs.berkeley.edu/Research/Projects/Data July 21,2010

[13] Huang M C Y,et al. Monolithic Piezo-Electric Actuated MEMS Tunable VCSEL[OL]. https://buffy.eecs.berkeley.edu/PHP/resabs/resabs.php

[14] VISIT - Vertically Integrated Systems [OL]. http://www.visit.tu-berlin.de 18.08.09

[15] SUBTUNE [OL]. http://www.subtune.eu Last update 01/02/2010 11:33:17

[16] 基于人类视觉系统的超级计算机[EB]. http://it.zaobao.com/pages9/itech100917.shtml,2010-9-17

[17] NeuFlow-Supercomputer Can 'See' Well Enough To Navigate Roads [OL]. http://www.science20.com/news_articles/neuflow_supercomputer_can_see_well_enough_navigate_roads, 2010-9-24

[18] A Dataflow Computer[OL]. http://www.eng.yale.edu/elab/resear-ch/svision/svision.html

[19] Yann LeCun. http://yann.lecun.com/

(上接第 278 页)

[5] Plimpton S. Fast parallel algorithms for short-range molecular dynamics[J]. Journal of Computational Physics,1995,117(1): 1-19

[6] Allen M P,Tildesley D J. Computer Simulation of Liquids[M]. Oxford:Claredon Press,1994

[7] Allen F, et al . BlueGene: A vision for protein science using a petaflop computer[J]. IBM Systems Journal,2001,40(2):310-327

[8] Gara A,et al. Overview of the Blue Gene/L System Architecture [J]. IBM Journal of Research and Development,2005,49(2/3): 195-212

[9] Phillips C,et al. Scalable Molecular Dynamics with NAMD[J]. J. Comput. Chem. ,2005,26(16):1781-1802

[10] Humphrey W, Dalke A, Schulten K. VMD-Visual Molecular Dynamics[J]. J. Molec. Graphics,1996,14(1):33-38

[11] Yang Jue-kuan, Wang Yu-juan, Chen Yun-fei. GPU accelerated molecular dynamics simulation of thermal conductivities[J]. J. Comp. Phys. ,2007,221(1):799-804

[12] Liu Wei-guo,et al. Accelerating molecular dynamics simulations using Graphics Processing Units with CUDA[J]. Journal of Computational Physics,2008,179(9):634-641