基于GPU的稀疏矩阵向量乘优化

白洪涛 欧阳丹彤 李熙铭 李 亭 何丽莉
(吉林大学计算机科学与技术学院 长春 130012)
(吉林大学符号计算与知识工程教育部重点实验室 长春 130012)

摘 要 针对稀疏矩阵运算难以发挥图形处理器的强大运算能力的现状,基于图形处理器的统一计算架构,在线程映射、数据复用等方面研究了一系列并行计算优化方法,从而完成了一种行压缩存储表示下的稀疏矩阵向量乘并行算法。这些优化方法包括:(1)利用Warp内线程天然同步特性,Half-warp完成结果向量一个元素的计算;(2)取整读取数据,实现合并访问;(3)输入向量放入纹理存储器,数据复用;(4)申请分页锁定内存,加速数据传输;(5)使用共享存储器,加速数据存取。实验分析表明,提出的各种手段起到了优化的作用。与已有的CUDPP和SpMV library中的CSR-vector算法相比,本算法获得了更高的存储器带宽和浮点运算吞吐量;整体性能比CPU串行执行版本快了3倍以上。

关键词 稀疏矩阵,行压缩存储,图形处理器,统一计算架构,优化策略
中图法分类号 TP311 **文献标识码** A

Optimizing Sparse Matrix-vector Multiplication Based on GPU

BAI Hong-tao OUYANG Dan-tong LI Xi-ming LI Ting HE Li-li
(College of Computer Science and Technology, Jilin University, Changchun 130012, China)
(Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China)

Abstract Sparse matrix computations present additional challenges for harnessing the potential of modern graphics processing unit(GPU) for general-purpose computing. We investigated various optimizations on thread-mapping, data reuse etc. and a parallel Sparse Matrix-Vector multiplication(SpMV) on GPU with compute unified device architecture (CUDA) was proposed under compressed sparse row(CSR) structure afterwards. The optimizations include: (1) exploiting each element using half-warp threads, which synchronize free within one warp; (2) making up integer address to achieve coalesced accesses; (3) data reuse through reading from texture vector resides in; (4) data transfer using page-locked memory; (5) reading results in shared memory. We compared the performance of our approach with that of efficient parallel SpMV implementations such as (1) the one from NVIDIA's CUDPP library and (2) the one from NVIDIA's SpMV library. Our approach outperforms two both in memory bandwidth and GFLOPS. In addition, the total performance of our approach is three times greater than that of a CPU counterpart.

Keywords Sparse matrix, Compressed sparse row, Graphics processing unit, Compute unified device architecture, Optimizations

现实中大量问题可用稀疏矩阵表示。稀疏矩阵向量乘(Sparse matrix-vector multiplication, SpMV)运算广泛地应用于大规模线性求解系统和求解矩阵特征值等问题[1],尤其在迭代方法中,SpMV成为影响算法性能的关键步骤。然而,SpMV是典型的存储器瓶颈类运算,即运算/访存比很低,运算器严重不饱和,难以达到高浮点运算吞吐量。SpMV具有本质的并行性,利用现代多处理器平台研究并行SpMV是提高其性能的可行方向之一。

现今,图形处理器(Graphics Processing Unit, GPU)成为个人计算机的标准配置。传统上,GPU的主要任务是图形图像的绘制和渲染。随着其可编程接口的开放以及高级绘制语言[2]的普及,GPU以其强大的并行处理能力和极高的存储器带宽向通用计算领域扩展。在模式识别、小波变换、计算智能和实时仿真等方面都取得了很好的效果[3]。GPU将成为高性能计算领域极具性价比的一个分支。

本文基于NVIDIA GPU的统一计算架构CUDA(Com-

到稿日期:2009-09-29 返修日期:2010-01-19 本文受国家自然科学基金重大项目基金(60496320,60496321),国家自然科学基金(60973089,60773097,60873148),吉林省科技发展计划项目基金(20060532,20080107),欧盟合作项目(155776-EM-1-2009-1-IT-ERAMUNDUS-ECW-L12)资助。

白洪涛(1975—),男,博士生,主要研究方向为高性能计算、基于模型的验证等,E-mail:baihongtao@263.net;欧阳丹彤(1968—),女,教授,博士生导师,主要研究方向为基于模型的诊断和定理机器证明、并行计算等;李熙铭(1986—),男,硕士生,主要研究方向为并行程序设计与实现;李 亭(1985—),女,硕士生,主要研究方向为并行程序设计与实现;何丽莉(1976—),女,博士,主要研究方向为智能计算与无线传感器网络等。

pute Unified Device Architecture)的特殊体系结构，在线程映射、数据复用等方面研究了一系列优化方法，从而完成了一种行压缩存储(compressed sparse row，CSR)表示下的稀疏矩阵向量乘并行算法。这些优化方法包括：(1)利用 Warp 内线程天然同步特性，Half-warp 完成结果向量一个元素的计算；(2)取整读取数据，实现合并访问；(3)输入向量放入纹理存储器，数据复用；(4)申请分页锁定内存，加速数据传输；(5)使用共享存储器，加速数据存取。实验分析表明，本文的各种手段起到了优化的作用。与已有的 CUDPP[4] 和 SpMV library[5] 中的 CSR-vector 算法相比，本文算法获得了更高的带宽和浮点运算吞吐量；整体性能比 CPU 串行执行版本快了 3 倍以上。

1 相关工作

存储器访问(gather or scatter)是影响 SpMV 效率的重要方面，而多核架构使存储器瓶颈的问题更加突出。总体来说，多核架构可以分为通用和专用两类体系结构。在通用架构，如 AMD Dual-core，Intel Quad core 平台上，主要手段是将局部数据放入 Cache 和寄存器(Register)中，调度算法的优劣影响算法的性能[6-8]。而本文所基于的专用结构 GPU 具有多级存储器体系[9]，需要根据问题的特征设计不同的优化策略，才能发挥 GPU 存储器高带宽的优势。

稀疏矩阵有多种存储格式，文献[5]给出了包括本文使用的 CSR 在内的 6 种存储方式，结果显示对于没有规律的数据，CSR 具有一定的优势。我们研究的矩阵整体上是稀疏的，但是可能存在较小的稠密子矩阵，这些密集子块有助于提高数据重用性。文献[10]对如何抽取一致尺寸的、非一致尺寸的、规则排列的和非规则排列的稠密子矩阵都进行了研究，把抽取的结果整块放入 Register 加速。此外，数据运行时排序也是提升性能的手段之一，Strout 等提出了一个该策略的框架[11]，这需要额外的计算，是否能够带来整体性能的提高还有待验证。

总体上，GPU 架构不同于基于高速缓存的通用架构，具有大规模并行性，即同一时刻有大量的线程处于活动状态，通过有效的线程调度隐藏全局存储器的高延迟访问。因此，细粒度线程级并行更适合 GPU。Buatois 等开发了基于 AMD CTM(close to the Meta)环境的稀疏线性求解器[12]，然而 AMD 的 GPU 维持传统的向量处理器架构(R,G,B,A)，与本文的 CUDA 标量架构有很大的区别。

2 CSR 存储格式

通常用 $m\times n$ 数组来存放 $m\times n$ 的稠密矩阵(m 行 n 列)，但这对稀疏矩阵而言效率很低。研究高效的稀疏矩阵存储方式，不仅可节省存储单元，还能够减少计算时间。目前稀疏矩阵尚无一种通用的最佳数据结构，不同的数据结构适合不同的变换操作和不同的实现方法。表现某些现实问题的矩阵有明显的特征，如主对角线对称。但绝大多数实际问题只具有稀疏的性质，矩阵本身没有规律。

CSR 是存储稀疏矩阵的有效方式之一。对于具有 q 个非零元素的稀疏矩阵，应用 CSR 格式存储时，使用 3 个数组：一个 $q\times 1$ 维的值数组 Data，它按行序分成了 m 个段；一个 $q\times 1$ 维的列下标数组 Indices；一个 $(m+1)\times 1$ 维的数组 Ptr，该数组中的元素指向各段中首元素在稀疏矩阵中的顺序号。图 1 给出了矩阵在 CSR 结构下的存储示例。

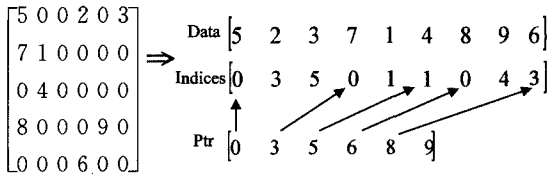

图 1 CSR 结构存储示例

与稀疏矩阵的其他存储格式相比，CSR 进行了行压缩，具有最佳的空间效率，同时能够方便地计算出第 i 行非零元素的个数($Ptr[i+1]-Ptr[i]$)。本文即采用该格式完成 SpMV。

3 优化策略

现代处理器在单个芯片内集成了或多或少的多个处理核心，并通过并行流水来提高处理器理论上的最高运算能力，未来会延续这一趋势。如 NVIDIA GTX 280 的 GPU 理论上的峰值性能约为 933 GFLOPS，全局存储器带宽的峰值超过 141 GBPS。这一方面为程序的高性能提供了条件，另一方面众多核心的并行化也使得内存器壁垒越来越突出。尤其是对本文的 SpMV 计算而言，数据从存储器取出后基本上只做一次计算，无法发挥缓存的效力。另外，由于 GPU 的 SIMT(Single Instruction Multiple Thread)架构，多级存储器体系模型及尺寸、访问等诸多限制使得基于 GPU 的内存瓶颈类算法困难重重。只有将计算本身的数据结构、计算特征等与 GPU 的线程映射、数据访问结合在一起，才有望获得较高的性能。由此，针对 NVIDIA 的 CUDA 研究了如下若干 SpMV 的优化策略。

3.1 线程映射

根据 SpMV 计算行无关的特性，并行 SpMV 最自然的想法是结果向量的每个元素由一个线程计算，即每个线程负责将稀疏矩阵的一行与输入向量相乘。这种方式较适合比较少的处理器核心且线程独立调度的并行计算平台。而对于 CUDA 的体系架构，SM(Stream Multiprocessors)将线程块划分为多个 Warp，以 Warp 为执行单元，同一 Warp 内线程自然同步，如图 2 所示。若处于一个 Warp 内线程对应处理的各行非零元素数目差异较大，则会造成诸多线程计算资源的空转等待，即条件分支。此外，GPU 需要调度大量线程才能有效掩盖全局存储器存取延迟，该策略在处理较小规模的矩阵(m 值)时更不具优势。文献[5]的 CSR(scalar)的实验也证实了该策略是低效的。

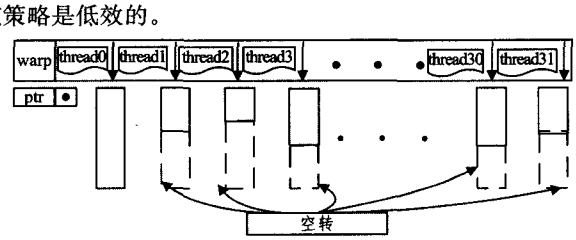

图 2 自然对应方式产生的条件分支

利用块内线程可同步的特性，一个线程块(Block)对应一行是一种较为适合 GPU 的模式。但是，通常一个 Block 也要含有诸多线程(256 是推荐设置)，当一行的非零元素个数不能被 Block 内线程总数整除时，也会有很大的概率导致若干线程无事可做；线程同步的额外开销也是该模式的劣势之一。

文献[5]的 CSR(vector)利用 Warp 内线程天然同步的特性，尝试了一个 Warp 计算一行(对应一个结果矩阵元素的计算)，每个 Block 计算 Block/Warp_Size 行。如图 3 所示，一个 Warp 有 32 个线程，在每行非零元素个数较少的时候仍然存在很多空转。

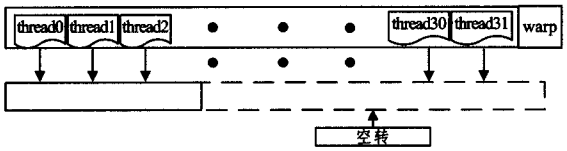

图 3　每行非零元素较少时的空转现象

在上述策略的基础上，结合合并访问的需要，以 Half-warp 为单位完成稀疏矩阵一行与输入向量相乘的计算，进一步缩减每行需要的线程数目，降低线程横向空转的比例。图 4 给出了本文的策略。显然，当非零元个数在相邻行差异较大时，自然对应方式的缺陷则再次出现，空转无法绝对避免，但可以减少 Warp 内同步等待。文献[13]采用了与本文相同的策略。

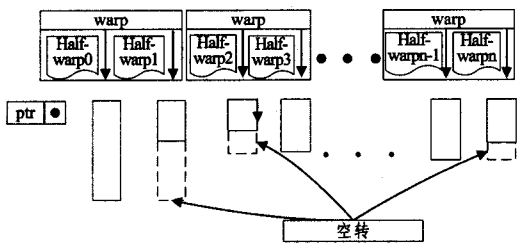

图 4　Half-warp 对应一行产生的条件分支

3.2　数据访问

首地址对齐和合并访问是优化 GPU 数据访问的重要方面，尤其是针对计算能力 1.2 以下的设备而言，因为首地址没有对齐而造成的非合并访问会严重影响整体性能。本文采用取整提取的方法，即将前一行的尾巴当作这一行的开始，多余的部分只提取、不计算，如图 5 所示。这与文献[13]的两阶段处理方法有所不同。

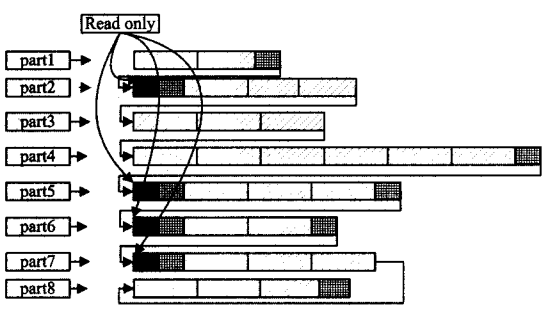

图 5　存储器对齐示例

合并访问是 Half-warp 内线程一起访问存储器，相邻线程访问相邻的数据，数据在存储器的一个段上，才有最佳的效率。如此，我们将数据以行优先的自然序存储，以满足合并访问的要求。

3.3　数据复用

GPU 提供了多层次存储器体系，不同的存储器分级管理。它们如下：(1)片外的全局存储器；(2)片外的局部存储器；(3)片上的共享存储器；(4)片外的常量存储器，拥有片上的 Cache；(5)片外的纹理存储器，拥有片上的 Cache；(6)本地 Register。

由于全局存储器的高延迟(200 个时钟周期)，应尽量减少全局存储器的访问。将数据先读入共享存储器复用，以提高访问存储器的性能。Ptr 的数据重用如图 6 所示。

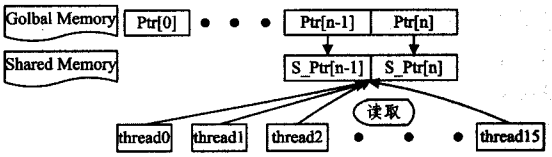

图 6　Ptr 数据重用

最后将输入向量放入纹理存储器，通过纹理 Cache 提升性能。

3.4　数据传输

CPU 和 GPU 之间的传输是基于 GPU 通用计算的额外开销。使用分页锁定的传输方式申请一块 CPU 内存，再将其拷贝到显存，这在一定程度上减少了数据传输的时间。然而，矩阵规模增大到一定程度时，锁定了太多的内存，同样会影响 GPU 程序(Kernel)的执行时间。实验部分我们将对此进行分析。

3.5　其他策略

・减少 Register 使用量，提高处理器占用率

优化前一个线程的 Register 使用量是 12 个，occupancy 仅为 0.667，即三分之二。经过分析，还有减少 Register 的余地。首先，将每个部分需要的计算量 part_size 作为参数传入 Kernel，尽量避免指令周期较长的除运算，减少了 1 个 Register 的使用。其次，乘积加和的 reduction 代码做如图 7、图 8 所示的改动，减少了 3 个 if 条件判断语句，同时减少了 1 个 Register 的使用。

```
if(threadIdx.x<8) shared_result[threadIdx.x])+=shared_result
[threadIdx.x+8];
if(threadIdx.x<4) shared_result[threadIdx.x])+=shared_result
[threadIdx.x+4];
if(threadIdx.x<2) shared_result[threadIdx.x])+=shared_result
[threadIdx.x+2];
if(threadIdx.x<1) shared_result[threadIdx.x])+=shared_result
[threadIdx.x+1];
```

图 7　修改前的 reduction 代码

```
if(threadIdx.x<8)
{
  shared_result[threadIdx.y][threadIdx.x]+=shared_result[threa-
dIdx.y][threadIdx.x+8];
  shared_result[threadIdx.y][threadIdx.x]+=shared_result[threa-
dIdx.y][threadIdx.x+4];
  shared_result[threadIdx.y][threadIdx.x]+=shared_result[threa-
dIdx.y][threadIdx.x+2];
  shared_result[threadIdx.y][threadIdx.x]+=shared_result[threa-
dIdx.y][threadIdx.x+1];
}
```

图 8　修改后的 reduction 代码

至此 Register 的使用量减少为 10 个。从 Profile 可以看到，多处理器的使用率 occupancy 达到 1。

・使用内部函数和位运算

Kernel 中尽量使用 CUDA 提供的标准数学函数，使用位运算代替指令周期较长的除法运算。

4 实验与分析

实验是在 CPU 为 AMD Athlon Dual Core Processor 3600+，GPU 是 Geforce 8800GTX(CUDA 2.0)，2GB 主存的环境中进行的，本文方法命名为 CUDA SpMV。稀疏矩阵来源于线性系统的标准测试用例[14]。

4.1 优化策略效果

线程映射和数据访问是 GPU 优化的最重要的两个方面。首先，给出对齐访问策略的有效性。如表 1 所列，Profiler 统计显示本文方法大幅度地提高了合并访问的数量。

表 1 数据对齐前后合并存取量的变化

	bcsstk31	bcsstk30	Fidapm37	fidap011
未对齐	842	1326	798	1196
对齐	18197	22678	14670	21788

图 9、图 10 给出了 Warp，Half-warp 及数据复用所带来的带宽和浮点吞吐量的提升，显示了本文各种策略的有效性，其中对齐访问和高速存储器对性能影响更大。

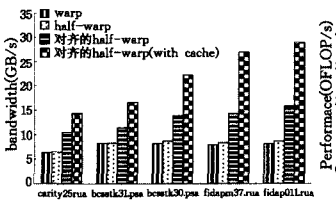

图 9 不同策略下带宽对比

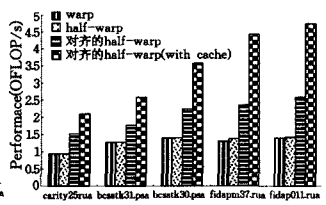

图 10 不同策略下浮点性能对比

4.2 带宽与浮点性能

CUDPP 和 SpMV library 是来自 NVIDIA 的最新的基于 CUDA 的 SpMV，IBM 研究院的方法获得了与 SpMV library 相近的结果[13]。本节将 CUDA SpMV 与 CUDPP 和 SpMV library 中的 CSR(vector)，ELL 和 HYB 进行比较，如图 11、图 12 所示。

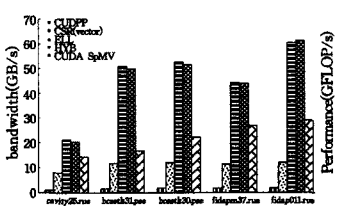

图 11 带宽横向对比

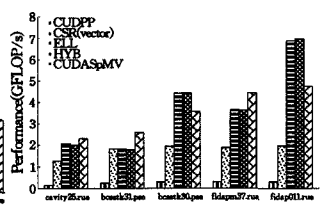

图 12 浮点性能横向对比

从图中可以看出，CUDA SpMV 在带宽和浮点性能上全面超越了 CUDPP 和 CSR(Vector)；HYB 和 ELL 在带宽上比 CUDA SpMV 高的原因是它比 CSR 存储了更多的数据。在实际浮点性能上，三者在不同的稀疏矩阵实例各有优势。总体上，在同样存储格式 CSR 下，本文的方法是最高效的。

4.3 加速比

本小节我们与 CPU 的串行执行的结果进行对比，图 13 给出了计算部分的加速比，图 14 给出了整体性能的加速比。

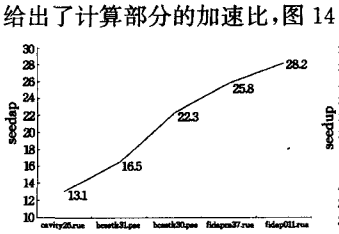

图 13 计算部分加速比

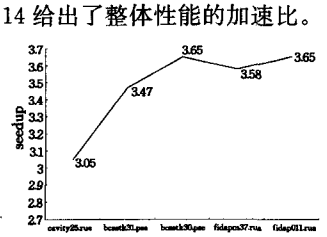

图 14 整体加速比

从上图中可以看出，随着矩阵规模和非零元素个数的增大，计算部分的加速比也随之增大，但 CPU 与 GPU 的数据传输时间也随之增大。图 15 特别给出了数据传输与计算时间的对比。

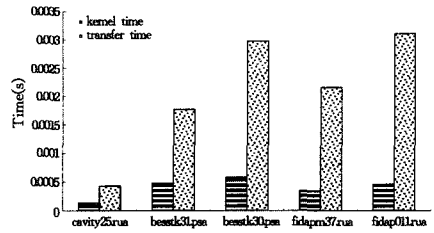

图 15 计算与数据传输时间对比

从图中可以看出数据的传输时间在整体时间中占了绝对的比重，SpMV CUDA 用于迭代计算则可以降低数据传输的影响。

Williams 等的研究显示[15]，基于 GPU 的 SpMV 与多核 CPU 上并行 SpMV(通常多线程)相比更具优势，本文不再重复该项比较。

结束语 本文以有代表性的存储器瓶颈类算法 SpMV 为例，将问题的特征与 GPU 的特殊体系结合在一起，研究了稀疏矩阵存储格式 CSR 下的若干 GPU 运算优化方法。在 Geforce 8800 GTX 平台上评估了本方法的有效性。稠密子矩阵识别和运行时数据重排等策略是我们进一步的优化方向，同时将基于 CUDA 的 SpMV 用于线性规划问题迭代求解，将是一个典型的应用。

参 考 文 献

[1] Saad Y. Iterative methods for sparse linear systems[M]. Society for Industrial Mathematics, 2003

[2] Foley T, Houston M, Hanrahan P. Efficient partitioning of fragment shaders for multiple-output hardware[C]// Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware. Grenoble, France, Eurographics Association, 2004: 45-53

[3] 吴恩华，柳有权. 基于图形处理器(GPU)的通用计算[J]. 计算机辅助设计与图形学学报，2004，16(5)：601-612

[4] CUDPP: CUDA data parallel primitives library[OL]. http://www.gpgpu.org/developer/cudpp/

[5] Bell N, Garland M. Efficient sparse matrix-vector multiplication on CUDA[R]. NVIDIA Technical Report NVR-2008-004. Dec. 2008

[6] Im E J, Yelick K A, Vuduc R. Sparsity: Framework for optimizing sparse matrix-vector multiply[J]. International Journal of High Performance Computing Applications, 2004, 18(1): 135-158

[7] Mellor C J, Garvin J. Optimizing sparse matrix-vector product computations using unroll and jam[J]. International Journal of High Performance Computation Application, 2004, 18(2): 225-236

[8] Nishtal A R, Vuduc R, Demmel J, et al. When cache blocking sparse matrix vector multiply works and why[J]. Applicable Algebra in Engineering, Communications and Computing, 2007, 18(3): 297-311

(下转第 181 页)

也对其产生了一定的影响，比如，在第 1200 个时间间隔处，多维 Hölder 指数的值就受到空闲的交换空间明显的影响。因此，如果能预测多维 Hölder 指数，便能够有效地预测系统资源动荡的情况，从而达到预测软件衰退的目的。

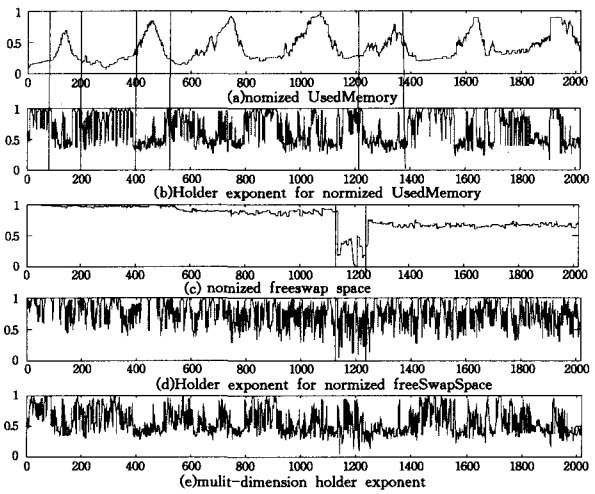

图 5　归一化参数序列及其对应的 Hölder 指数

ARMA 模型能够有效地分析平稳性数据序列的相关性。下面将利用 ARMA 对多维 Hölder 指数进行预测分析。首先建立 ARMA 模型，为了减少计算量，这里采用文献[5]中的 ARMA(2,1)模型来进行预测。由最小二乘估计方法解出相应的参数值：$\phi_1=0.52797$，$\phi_2=0.33075$，$\theta_1=0.11287$，$\delta_\varepsilon^2(P)=0.44256$。满足 $\phi_1+\phi_2<1$，$\phi_2-\phi_1<1$，$|\phi_2|<1$，根据平稳性判定准则，可判定它是一个平稳时间序列，说明前面的判断是正确的，于是拟合的 ARMA(2,1)模型为：

$$H_t=0.52797H_{t-1}+0.33075H_{t-2}+\varepsilon_t-0.11287\varepsilon_{t-1} \quad (4)$$

针对上述拟合模型，利用模型递推法来实现 ARMA(2,1)模型 1 步预测。令 $\varepsilon_{t-3}=0$，那么如果要实现某点的 1 步预测值，只要知道之前的 4 个观测值即可：

$$H_t(1)=0.52797H_{t-1}+0.33075H_{t-2}-0.11287\varepsilon_t \quad (5)$$

由于 ARMA 为短相关模型，因此 1 步预测的精度优于多步预测。我们利用前 7 天的资源参数数据的多维 Hölder 指数，对第 8 天第一个小时内的指数进行 1 步预测。如图 6 所示，可以看出，指数的值呈下降趋势，表示对应的系统资源参数的动荡指数将会加剧，这与先前的分析以及实际应用中系统资源随着长时间的持续运行会出现耗尽的情况是一致的。若对指数设定阈值，当预测指数将下降，而且将在可预见时间内达到指定阈值时，能够提前给出软件衰退预警，以表明系统出现资源耗尽的征兆，需要实施预防性的软件自愈操作。至此达到了预测软件衰退的目的。

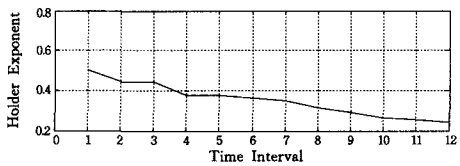

图 6　第 8 天第一个小时多维 Hölder 指数的预测值

结束语　本文给出了一种基于多重分形分析、定性与定量相结合分析软件衰退的方法。通过实证分析说明了与系统内存资源相关的两个参数(使用的物理内存、空闲的交换空间)是具有多重分形特性的，而且系统参数多重分形谱分布为认识参数的发展趋势提供了有价值的信息。同时，采用 ARMA(2,1)模型建模两个参数对应的多维 Hölder 指数，能够对多维 Hölder 指数短期内的数值进行较准确预测，能够估算短期内到达警戒阈值的时间，满足软件衰退预测需求，为制定软件恢复策略提供依据。

参 考 文 献

[1] Shereshevsky M, Cukic B, Crowel J, et al. Software Aging and Multifractality of Memory Resources[C]// Pceedings of DSN 2003. Los Alamitos, USA: IEEE Computer Society, 2003: 721-730

[2] Chen Xiu-E, Quan Quan, Jia Yun-Fei, et al. A Threshold Autoregressive Model for Software Aging[J]. Service-Oriented System Engineering, 2006: 34-40

[3] Vaidyanathan K, Trivedi K S. A Measurement-based Model for Estimation of Software Aging in Operational Software Systems [C]// Proceedings of 1999 International Symposium on Software Reliability Enginering. Los Alamitos, USA: IEEE Computer Society, 1999: 84-93

[4] Li L, Vaidyanathan K, Trivedi K S. An Approach for Estimation of Software Aging in a Web Server[C]// Proceedings of 2002 International Symposium on Empirical Software Engineering. Los Alamitos, USA: IEEE Computer Society, 2002: 91-100

[5] 邹柏贤，刘强. 基于 ARMA 模型的网络流量预测[J]. 计算机研究与发展，2002，39(12)：1645- 1652

[6] 徐建，张琨，刘凤玉. 基于分形的软件衰退预测[J]. 系统仿真学报，2007，19(3)：549-551

(上接第 171 页)

[9] NVIDIA CUDA[OL]. http://developer. nvidia. com/object/cuda. html

[10] Vuduc R W, Moon H J. Fast sparse matrix vector multiplication by exploiting variable block structure[C]// Proceedings of the International Conference on High-performance Computing and Communications. LNCS v3726. Sorrento, Italy, 2005: 807-816

[11] Strout M M, Carter L, Ferrante J. Compile-time composition of run-time data and iteration reorderings[J]. ACM SIGPLAN Notices, 2003, 38(5): 91-102

[12] Buatois L, Gaumon G, Levy B. Concurrent Number Cruncher: An efficient sparse linear solver on the GPU[C]// High Performance Computation Conference (HPCC). Spring Lecture Notes in Computer Sciences, LNCS v4282. 2007: 358-371

[13] Baskaran M M, Bordawekar R. Optimizing sparse matrix-vector multiplication on GPUs [R]. The Ohio State University, Columbus, USA and IBM TJ Watson Research Center Hawthorne, NY, USA, 2009

[14] A visual repository of test data for use in comparative studies of algorithms for numerical linear algebra[OL]. ftp://math. nist. gov/pub/MatrixMarket, 2007

[15] Williams S, Oliker L, Vuduc R, et al. Optimization of sparse matrix-vector multiplication on emerging multicore platforms[J]. Parallel Computing, 2009, 35(3): 178-194