Interaction and Multiscale Mechanics, *Vol. 6, No. 2 (2013) 197-210*
DOI: http://dx.doi.org/10.12989/imm.2013.6.2.197

Meshfree/GFEM in hardware-efficiency prospective

Rong Tian*

Institute of Computing Technology, Chinese Academy of Sciences, Kexueyuan Nanlu 6, Haidian, Beijing 100190, China

(Received March 16, 2013, Revised April 28, 2013, Accepted May 6, 2013)

Abstract. A fundamental trend of processor architecture evolving towards exaflops is fast increasing floating point performance (so-called "free" flops) accompanied by much slowly increasing memory and network bandwidth. In order to fully enjoy the "free" flops, a numerical algorithm of PDEs should request more flops per byte or increase arithmetic intensity. A meshfree/GFEM approximation can be the class of the algorithm. It is shown in a GFEM without extra dof that the kind of approximation takes advantages of the high performance of manycore GPUs by a high accuracy of approximation; the "expensive" method is found to be reversely hardware-efficient on the emerging architecture of manycore.

Keywords: meshfree; GFEM; manycore; co-design; exascale computing

1. Introduction

The work is motivated by the challenge in fully utilizing high floating point performance on the emerging manycore architecture. A fundamental trend of computer architecture evolving towards exaflops (10^{18} floating point operations (flops) per second) is the fast increasing compute power (the so-called "free" flops) accompanied by much slowly increasing memory and network bandwidth. Numerical simulation is seeing an unbalanced increase in the floating point performance (which increases fast) and the memory bandwidth (which increases much slowly). In other words, numerical simulation is likely to be subject to a memory constraint instead of a floating point performance constraint. As a result, a numerical method of PDEs should be designed to request more flops per unit memory access so that the "free flops" are not "wasted' (DOE Exascale Initiative Roadmap 2009, DOE Office of ASCAC 2010, Tian and Sun 2013, Tian 2012).

A meshfree method is able to increase arithmetic intensity for the same number of dofs by increasing the influence radius or by increasing the accuracy of approximation and therefore it may bear a potential in utilizing the redundant floating point capability of the emerging manycore processor. This idea is tried out on a GFEM without extra dof in this paper.

In the next section, we introduce the "performance gap" of the today's supercomputers, which is a huge performance difference between real Scientific and Engineering (S&E) applications and the LINPACK benchmark. In Section 3, by remarking on the trend of hardware change, we point out that the performance gap might be further widen if numerical algorithms do not change

*Corresponding author, Professor, E-mail: rongtian@ncic.ac.cn accordingly to catch up with the change of the computer architecture. Following the trend of the hardware change, we believe that a "good" numerical algorithm would be differently defined and this is the main content of Section 4. In Section 5, as an example demonstrating how to design a hardware-efficient numerical algorithm, we introduce a GFEM without extra dof and a numerical example is provided to demonstrate the method's capability of delivering a different order of accuracy and convergence for the same number of dofs—a unique feature common to many meshfree methods. In Section 6 we test the floating point efficiency of the new GFEM on an nVidia GPU, followed by the conclusions of the paper.

2. "Performance gap" of today's computers

In the HPC community, domain application experts commonly acknowledge that even a finite element code delivers a nearly linear speedup, the percent of the peak performance of a computer that the code can really utilize is rather small. For an implicit method, the percent is rarely beyond 20%. As an example, we estimate in Table 1 the theoretical upper bound of the performance of a matrix vector (MV) multiplication—the core of the most of iterative methods—in double precision on AMD Opteron 6274 (Interlagos) CPU, Tesla k20x GPU and the fastest supercomputer Titan (which is the #1 fast supercomputer in the time of the paper publication). From Table 1 it is seen that the efficiency on the hardware is not beyond 10%, which says that only less than 10% of the floating point peak performance can be practically utilized, leaving 90+% underutilized in the MV computation!

On the other hand, the efficiency obtained for the linear algebraic package (LINPACK) benchmark test (Top500 website) is more than 80% in general. For the K-computer, the LINPACK test even delivers 93% efficiency. The MV kernel represents the core characteristics of a majority of S&E applications. Clearly, there exists a huge gap between the real S&E application performance and the "nominal" bench mark performance (Fig. 1) (Tian and Sun 2013, Tian 2012).

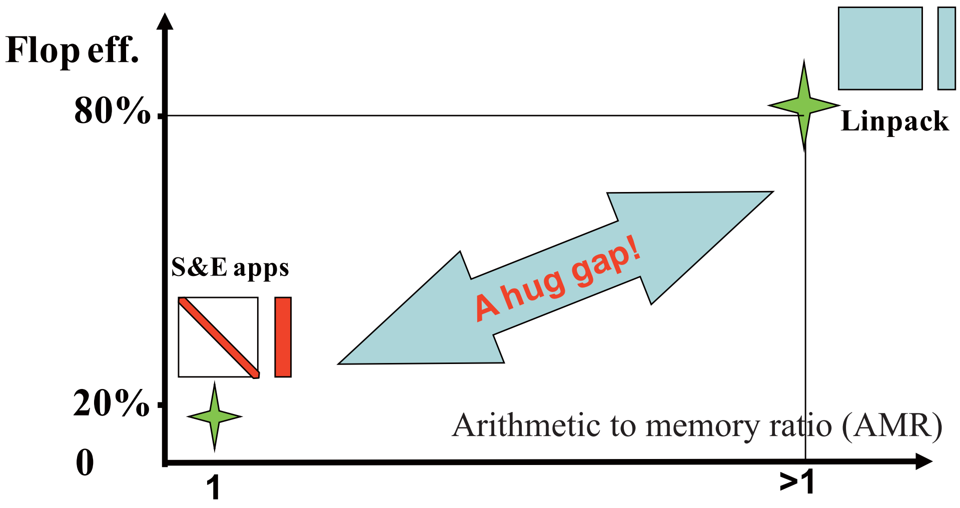

Fig. 1 Performance gap

Table 1 Theoretical upper bound of efficiency of matrix-vector (MV) multiplication (Arithmetic to memory ratio (AMR): 2:1) in double precision

Hardware	AMD Opteron 6274	GPU(Tesla k20x)	Titan
Performance(Gflop/s)	141	1311	1452
Memory bandwidth(GB/s)	52	180	232
Upper bound of efficiency	10%	3.4%	4%

The reason behind the performance gap is not complicated. A large portion of the S&E applications is to solve a system of linear equations. The corresponding numerical algorithm is mostly an iterative method. The iterative method is finally reduced to sparse matrix vector multiplication (spMV). The theoretical upper bound of the "arithmetic to memory ratio" (AMR) (measured in flop/word) of spMV is only 2:1, which is typically a memory intensive algorithm (computing is memory bandwidth bound). However, the LINPACK benchmark test uses a direct method to solve the system of linear equations. Its core algorithm is matrix multiplication (DGEMM) and the AMR is n:1, where n is the size of the sub-matrix, which features computing intensive. The AMRs of the algorithms of the real S&E application and the LINPACK test are fundamentally different. It is the fundamental difference in algorithm that determines the huge gap of a computer's performance in the real S&E application and in the LINPACK benchmark test.

3. Tomorrow's hardware change

Fig. 2 shows the performance of two nVidia GPUs, Tesla C2050 and Tesla C1060 (nVidia website). The specifications of the two GPUs are list in Table 2. The left-handed side figure is the results of the matrix multiplication test using the DGEMM (Double precision General Matrix Multiply) subroutine of the cuBLAS3.1 library, which is also the core algorithm of the LINPACK benchmark test. The right-handed side is tested using 13 real S&E applications (which are mostly finite element/difference methods). The core algorithm of the S&E applications is the SParse Matrix-Vector multiply operation, spMV. The tests are originally for a purpose of highlighting good performance of the manycore GPUs over the multicore CPU. With the same data, made possible is a comparison between the GPUs representing the emerging hardware architecture.

The floating point peak performance of C2050 is 515 Gflop/s in double precision, while that of C1060 is 78 Gflop/s. The former is 6.6 times fast. If asking "*which hardware is better*", a quick answer very likely goes the former quickly because the experience of the PC times tells us that "the faster the better".

Now let us focus on the performance difference of the two GPUs (the green and yellow lines in each figure). From Fig. 2, it is seen that the performance of C2050 and C1060 for DGEMM (the left figure) are significantly different, whereas the performance of the two GPUs are quite close for spMV (the right figure). C2050 is about 2 times fast compared with C1060 for DGEMM whereas only 40% faster than C1060 for spMV. The performance of the same hardware is significantly different for the different algorithms. The difference should lie in the following fact: DGEMM is computing intensive, whose performance is determined by the floating point performance of the hardware, whereas spMV is memory intensive, whose performance is determined by the memory bandwidth of the hardware. The memory bandwidth of C2050 (144 GB/s) is 46% wider than C1060 (102 GB/s). This poses a hardware upper limit on the performance gain of spMV on C2050.

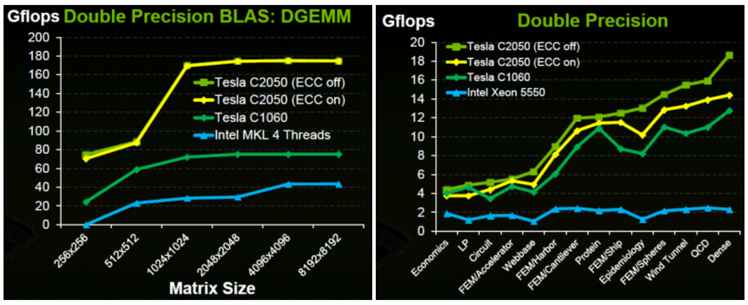

Fig. 2 Performance tests of Tesla C2050 and Tesla C1060 GPUs (source: nVidia website)

Table 2 nVidia Tesla C-series GPUs: C2050 and C1060

GPU	Power	Memory	Performance	Memory bandwidth
C1060	188W	4GB	78Gflop/s	102GB/s
C2050	247W	3GB	515Gflop/s	144GB/s

Therefore, the performance improvement of C2050 over C1060 for spMV does not exceed 46%. This explains the 40% performance improvement for the real S&E applications, although C2050 is 6.6 times fast compared with C1060. Taking into considerations of power consumption and price, C1060 is not too bad. In terms of floating point efficiency or power efficiency (the percent of the floating point peak performance can be effectively used), C1060 is 14% (11/78) efficient whereas C2050 only 3% (16/515). C1060, the product out of date, is reversely more efficient than the newer product C2050.

Obviously, the *memory intensive* S&E applications does not simply benefit from the pure increase in the floating point peak performance. It is the balance between the floating point performance and the memory bandwidth that predetermines the performance gain of the real S&E applications. The key algorithm of the most S&E applications is largely memory intensive. If no change is made to the key numerical algorithm, the fast increasing floating point capability is increasingly redundant and the waste of it would be unavoidable. Simply to increase the number of cores is not necessarily a good news to the most of S&E applications and reversely it tends to widen the existing performance gap. A revisit to the key numerical algorithm is critical to catch up with the hardware change.

4. “Good” algorithms on emerging architectures

In the PC times or the single core times, we usually assume that flops are expensive. As such the “best” practice that we have been following is to trade frequent memory accesses for reduced/saved flops and the philosophy of code and algorithm optimization then also is mainly aiming at reducing the number of flops. However, following the current trend of hardware change, the tomorrow’s S&E applications would unavoidably face a higher “performance to memory bandwidth ratio”. GPUs are one of typical examples. Due to the rapid increase in compute power and the relatively slowly increase in the capability of data movement, flops are becoming the next round of “free lunch”, whereas the data movement becomes the new bottleneck. Hence, from the viewpoint of hardware efficiency and energy effectiveness, the more flops per unit memory access, the merrier. This is in perfect contrast with the programming habit and the thinking way of algorithm development in the past.

In short, the tomorrow's "good" algorithm might be differently defined.

The algorithm that was deemed to be expensive in the PC or the single core times might be reversely a good candidate on the emerging computer architecture. For the future's exascale system, the S&E application should adopt the algorithm that can maximize the number of flops per unit memory access to take advantage of the "free flops". To domain experts, this implies that they may have to re-visit the core numerical algorithm of their applications and to re-think its effectiveness on the emerging computer architecture.

4.1 A new prospective on meshfree methods

For a given mesh, a meshfree approximation offers high accuracy for a large influence radius. Roughly speaking, the mesh or the number of dofs is related to memory requirement, whereas the influence radius is related to the number of flops. Therefore, when the influence radius increases, the AMR should also increase—for n points in the influence circle, each point is repeatedly used n times. This is to say that the arithmetic intensity in the meshfree approximation is tunable through changing the influence radius; this is a unique feature of the meshfree method over the standard FEM. In the standard FEM, the accuracy of approximation can only be improved by increasing nodes or dofs. By taking into considerations the facts and observations in Sections 2 and 4, we believe that the meshfree approximation may better fit the emerging manycore computer architecture.

Exactly, in the PC or the single core times, we value sparsity: the sparser the matrix, the more efficient; a high order method usually is deemed inefficient as it leads to a denser matrix problem. However, a moderately dense matrix means more arithmetic operations and potentially more computing intensive. Meanwhile, the dense matrix often is a sign of high accuracy of a numerical approximation. Therefore, the moderately dense matrix may become a combination of the hardware efficiency and the high numerical accuracy. This is a "win-win" strategy. As such, on the emerging manycore architecture we may value a numerical method which leads to the moderately dense matrix. This signals a philosophy shift in selecting, optimizing and developing a "good" numerical algorithm. As illustrated in Fig. 3 the meshfree and GFEM approximations, which are considered to be expensive, may deserve a re-visit from a prospective of hardware-efficiency.

5. The GFEM without extra dof: an example

The fantastic possibilities of the Partition of Unity (PU) approximation were first elaborated in the Partition of Unity Method (PUM) and the Partition of Unity Finite Element Method (PUFEM) by Melenk and Babuška (1996, 1997) (and earlier in Babuška *et al.* (1994)). The similar idea to the PUM was also introduced in the *hp*-cloud method by Duarte and Oden (1996, 1998). The finite element PU is extensively investigated in the PUFEM (Melenk and Babuška 1996), the generalized finite element method (GFEM) by Strouboulis *et al.* (2000a, 2000b, 2001, 2004, 2006a, 2006b, 2008) and the GFEM by Duarte *et al.* (2000, 2001).

The core of the GFEM is a PU approximation (in 1D)

$$u^h(x) = \sum_{i\in I} N_i u_i + \sum_{i\in I} N_i \sum_k \varphi_k^{P_i} a_{i(k)} \tag{1}$$

where $\sum_i N_i(x) \equiv 1$ form the PU, $\varphi_k^{\mathrm{P}_i}$ is the user-defined local function on patch P_i which is composed of elements surrounding node *i* and $a_{i(1)}$, $a_{i(2)}$, ... are the extra dofs of node *i*.

The number of dofs per node in the existing GFEM also varies with the order of local approximation and hence AMR cannot be improved. An extra-dof-free and linearly independent enrichment, which may lead to an improved AMR, is proposed in Tian (2013) and is briefed below.

5.1 A GFEM without extra dof

Let P_i^r be a patch composed of elements surrounding node *i*, where *r* denotes the size of the patch. The patch size is either the size of nodal support combining *m*≥1 layer(s) of elements surrounding node *i* on a regular mesh or simply the radius of an influence circle at node *i* on an arbitrary mesh. Node *i* is referred as "patch star" and index *i* is solely kept for the patch star and *j*

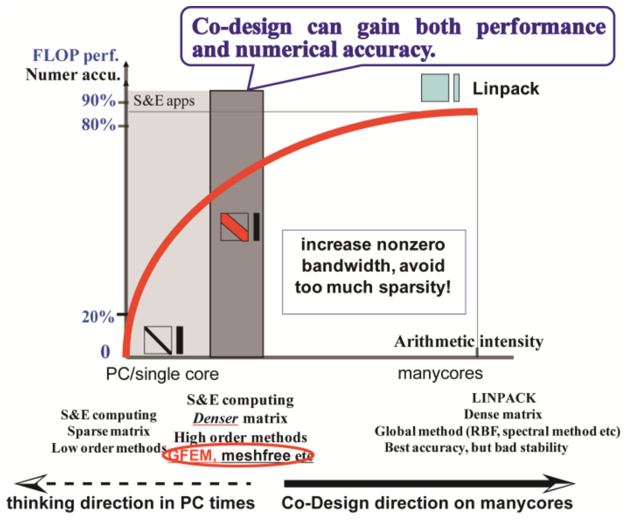
Co-design can gain both performance and numerical accuracy.
FLOP perf.
Numer accu.
90%
80%
20%
0
S&E apps
Linpack
increase nonzero bandwidth, avoid too much sparsity!
Arithmetic intensity
PC/single core
manycores
S&E computing
Sparse matrix
Low order methods
S&E computing
Denser matrix
High order methods
GFEM, meshfree etc
LINPACK
Dense matrix
Global method (RBF, spectral method etc)
Best accuracy, but bad stability
thinking direction in PC times
Co-Design direction on manycores

Fig. 3 "Good" algorithms on emerging architecture

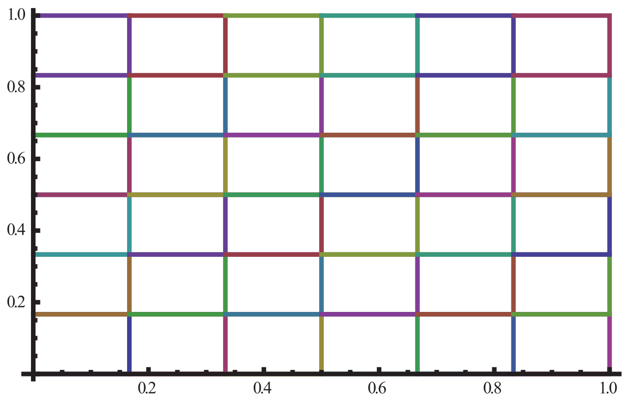

Fig. 4 Mesh for convergence tests

$(j \neq i)$ is used to denote any a non-patch star node on the patch. Node set $\{x_k | x_k \in \mathrm{P}_i^r\}$ is a collection of *all* the nodes on P_i^r .

Now let us construct an approximation of $u_i(x), x \in \mathrm{P}_i^r$, among the nodal values of $\{x_k | x_k \in \mathrm{P}_i^r\}$ local to P_i^r as

$$u_i(x) = \sum_{k=1}^{n_i} L_k^{\mathrm{P}_i^r}(x) u_k \ , x, x_k \in \mathrm{P}_i^r \tag{2}$$

where $L_k^{\mathrm{P}_i^r}(x)$ is the local function defined with regard to *k*-th node in the node set (which usually also forms a partition of unity, $\sum_{k=1}^{n_i} L_k^{\mathrm{P}_i^r}(x_i) \equiv 1$), n_i is the number of nodes on the patch and u_k is the conventional nodal unknown.

Use the approximation $u_i(x)$ as a local approximation at node *i* and directly substitute it for the nodal unknown u_i in the standard FEM approximation

$$u^h(x) = \sum_{i=1}^{N} N_i(x) u_i \tag{3}$$

where N_i is the standard FE shape function, a new approximation then is obtained as

$$u^h(x) = \sum_{i=1}^{N} \left(N_i \left(\sum_{k=1}^{n_i} L_k^{\mathrm{P}_i^r} u_k \right) \right) \tag{4}$$

where *N* is still the number of nodes on the mesh of the standard FEM (Eq. (3)) and u_k is still the nodal unknown of the standard FEM. Different from the standard FEM, one individual node appears repeatedly in n_i patch-wise local approximations in Eq. (4).

In approximation (4), corresponding to the n_i nodes on patch *i*, each node, u_i, has n_i local functions, which are respectively

$$\underbrace{N_i L_i^{\mathrm{P}_i^r} u_i}_{\text{patch star node}} \ , \ \underbrace{N_1 L_i^{\mathrm{P}_1^r} u_i,\ N_2 L_i^{\mathrm{P}_2^r} u_i,\ \cdots,\ N_{j,j\neq i} L_i^{\mathrm{P}_j^r} u_i, \cdots,\ N_{n_i} L_i^{\mathrm{P}_{n_i}^r} u_i}_{\text{non patch star nodes}} \tag{5}$$

where the superscript *k* in the standard FE shape function N_k , *k* = 1, 2, …*i*, …, *n_i*, is a local number (which is in contrast to the global numbering implied in Eqs. (3) and (4)), which is defined at the nodes of patch *i* and $L_i^{\mathrm{P}_k^r}$ is the local function constructed using the nodes (which are partially from patch *i*) from patch *k* with regard to node *i* (which is shared by patches *k* and *i*). Noted is that $L_i^{\mathrm{P}_k^r}$ and $L_k^{\mathrm{P}_i^r}$ is not equivalent.

By keeping in mind Eq. (5) and re-grouping the terms in Eq. (4) by u_i, we obtain

$$u^h(x) = \sum_{i=1}^{N} \left(\sum_{k=1}^{n_i} N_k L_i^{\mathrm{P}_k^r} \right) u_i = \sum_{i=1}^{N} \tilde{N}_i u_i \tag{6}$$

where n_i, the number of nodes on patch P_i^r in (2) now really means the number of patches containing node *i,* but the two numbers are the same and $\tilde{N}_i$ denotes the new shape function, which is a summation of n_i terms. This completes the basic idea of the work, which as seen above is quite straightforward.

Comparing Eq. (4) with Eq. (1), it is immediate that the form Eq. (4) is also a PU approximation. However, the new PU approximation has two unique features: (1) it does not include any extra dof (and accordingly it should be linearly independent); (2) the PU approximation interpolates as long as the patch-wise local approximation interpolates at patch star *i*, regardless of that it interpolates or not at the non-patch star nodes.

A large patch size *r* facilitates a high degree of $L^{\mathrm{P}_i^r}$ and hence a high order of the global approximation. This resembles the core feature of the existing GFEM. The new PU approximation does not include any extra dof and the total number of dofs to be solved is invariable with the local function $L^{\mathrm{P}_i^r}$. Both an interpolation and a *selectively interpolating* approximation can be used to construct the local approximation (Tian 2013). When an approximation (for example the moving least squares approximation (Lancaster and Salkauskas 1981) is used to construct the local approximation, *selectively interpolating* means the approximation should be enforced to pass the patch star, which is easy. Finally, the new GFEM always interpolates, no matter the local approximation is approximative or interpolative (Tian 2013).

5.2 Lagrange interpolation polynomial local approximation

On a structured mesh, the local approximation can be constructed by Lagrange interpolating polynomials. For patch size $r = ch$ (*h* is the mesh size), the local function $L_k^{\mathrm{P}_i^r}(x)$ can be expressed as Lagrange interpolating polynomials in *x* direction

$$L_k^{\mathrm{P}_i^{r_x}}(x) = \prod_{j=1,j\neq k}^{n_i} \frac{(\xi - \xi_j)}{(\xi_k - \xi_j)}, \quad \xi = \frac{x - x_i}{h} \tag{7}$$

where r_x is the patch size in *x*. In high dimensions the tensor product of the above one-dimensional Lagrange interpolating polynomial can be used. The order of $L_k^{\mathrm{P}_i^{r_x}}$ is solely determined by and increases with n_i.

The RBF and the Selectively Interpolating MLS can be used to construct a local approximation for an unstructured mesh of any dimension. Further detail of the new GFEM can be found in Tian (2013). In the study, main attention is paid to its floating point performance on the emerging architecture. The numerical performance of the new GFEM does resemble the existing GFEM; tests are provided to demonstrate its capability of offering a different order of accuracy and convergence for the same number of dofs.

5.3 Numerical performance of the new GFEM

The following benchmark problem is considered in 2D

$$\begin{aligned} \Delta\mathbf{u} &= -\mathbf{f} \quad in \quad \Omega \\ \mathbf{u} &= 0 \quad on \quad \partial\Omega \end{aligned} \tag{8}$$

on a regular domain $\Omega = [0,1]^2$. The exact solution is taken to be

$$u_x = u_y = xy(1-x)(1-y)\sin(x)\sin(y) \tag{9}$$

Errors are measured by the L$_2$ and energy norms defined respectively below

$$\|u\| = \left(\int_\Omega \left(u^h(x) - u(x)\right)^2 \mathrm{d}\Omega\right)^{\frac{1}{2}}, \quad \|e\| = \left(\int_\Omega \left(\nabla u^h(x) - \nabla u(x)\right)^2 \mathrm{d}\Omega\right)^{\frac{1}{2}} \tag{10}$$

The mesh is taken to be a Cartesian mesh that is refined uniformly for convergence tests (Fig. 4). The patch size is defined by a square nodal patch. In the following, the tensor product Lagrange interpolation polynomial local approximation is tested in Fig. 5. c+2 point element-wise Gaussian quadrature is used.

The new GFEM offers improved accuracy and elevated convergence for an enlarged patch size; the convergence rates in both L$_2$ and energy norms increase one order in general when the patch size increases 1h (refer to the rate number in the convergence plots). The convergence property resembles the existing GFEM. The most unique feature of the new GFEM is capable of offering different order of accuracy and convergence for the same mesh and the same number of dofs. Normally this is a feature offered by a finite difference method, not readily shared by a FEM. In contrast, in the existing GFEM accuracy and convergence can only be improved through increasing nodes or dofs or the both.

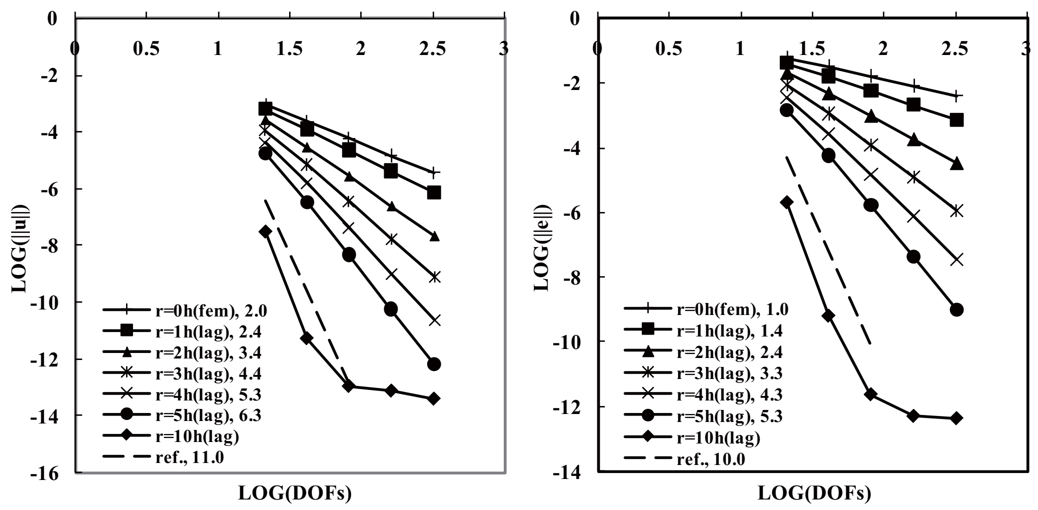

Fig. 5 Convergence test results. The tests show that the new GFEM is able to offer an elevated accuracy and convergence on the same mesh and the same number of dofs

Table 3 Test platform specification

GPU	nVidia GeForce GTX 285
CPU	Intel Xeon L5420
OS	CentOS release 5.4
CUDA	CUDA 4.0
CUSP	CUSP 0.2.0

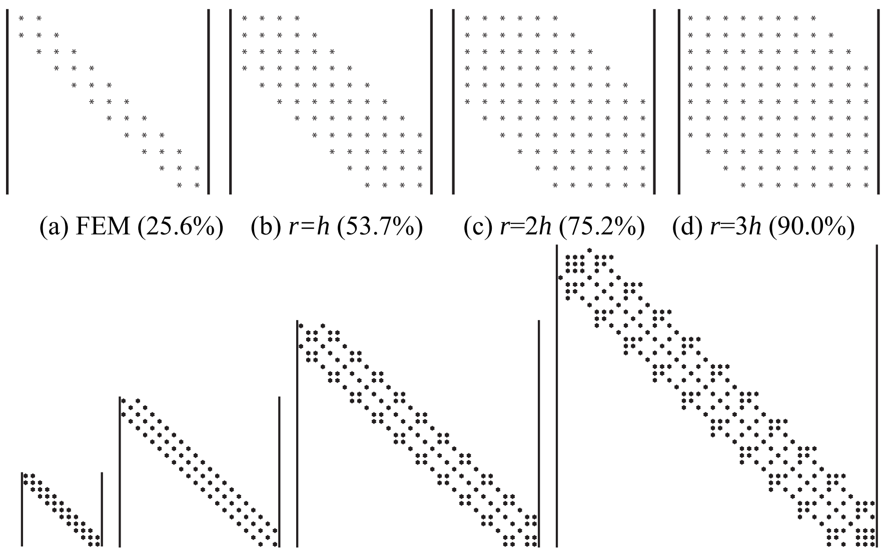

Fig. 6 Stiffness matrices on an 10-element mesh in 1D. (a)-(d) for the new GFEM, (e)-(h) for the existing GFEM, the number in bracket is the nonzeros percent, reflecting the sparsity of matrix. The existing GFEM shows nearly constant sparsity regardless of p while the new GFEM reduces the sparsity of matrix by increasing r

Traditionally, we are not prone to consider the method efficient because it leads to a large nonzero bandwidth. This is particularly true in the PC or the single core times. However, on the emerging architecture, in particular manycore processors, the tunable sparsity might be a different thing. Tests follow.

6. Floating point performance on emerging architecture

The GFEM with extra dof has been extensively studied in the contexts of numerical accuracy, convergence and stability in Tian (2013). Its hardware efficiency is investigated in the paper.

The following floating point performance test is focused on solving the system of linear equations associated with the new GFEM. Element assembly, as it is related to the definition of the local approximation, is excluded.

The global stiffness matrices of the new GFEM and the existing GFEM are drawn for comparison for an 10-element mesh in 1D for different patch sizes in Fig. 6. The sparsity of matrix is defined by the approximate ratio of nonzeros; the larger the ratio, the denser the matrix.

The differences in the global stiffness matrices between the two methods are that:

• as the order of local approximation increases, the global matrix does not vary in size in the new GFEM whereas it expands in the existing GFEM.

• as the order of local approximation increases, the matrix sparsity is reduced in the new GFEM whereas it rarely changes in the existing GFEM.

• The sparsity of the global matrix in the new GFEM is variable or *tunable* by varying patch size, *a common feature of many meshfree methods*.

The first test is to perform spMV. spMV is the kernel algorithm of an iterative equation solving method. Considering the banded shaped stiffness matrix shown in Fig. 6(a), we test the efficiency of spMV at a different nonzero bandwidth *d* on an nVidia GPU. Table 3 lists the software and hardware used in the tests. Here a large *d* corresponds to an approximation of a high order of accuracy and convergence. The five commonly used sparse matrix formats are considered (see Fig. 7). By varying the nonzero bandwidth of the matrix, we repeat spMV for the same matrix and vector and measure its performance in Gflop/s. The results are plotted in Fig. 7.

It is observed that as *d* increases, i.e., the stiffness matrix becoming denser or the order of approximation increasing, the performance of spMV increases regardless of the sparse matrix format. Since the large *d* also means an approximation of high order of accuracy and convergence, the tests show that on the manycore architecture the performance is gained by using the high order of approximation. This means the high order of approximation, i.e., the moderately large *d*, is more hardware-efficient than the lower order, i.e., the small *d*.

While the above test is simple, it is very instructive: a numerical method with tunable sparsity may offer a chance in shifting itself from “memory intensive” to *moderately* “computing intensive”. Next, we give a full test on this point with the new GFEM.

The 2D problem of Eq. (8) is solved by the new GFEM. The calculation is carried out on both the CPU (using a single core) and the GPU to test the method’s hardware-efficiency. The performance is measured in Gflop/s. Comparison is confined to the expense of solving the system of linear equations; the element stiffness assembly is out of consideration. The conjugate gradient method is used to solve the system of linear equations. The open source library CUSP (http://code.google.com/p/cusp-library/) is used for the conjugating gradient method on the GPU. The sparse matrix is stored in the Compressed Sparse Row (CSR) format (as the CSR format is generally used in practice).Test results are shown in Fig. 8.

First, we reduce the new GFEM to the standard FEM by letting r=0h and compare the sustained performance (Gflop/s) on the manycores of the GPU and the single core of the CPU. The performance data for the FEM on the CPU and the GPU are circled in the figure. It is observed

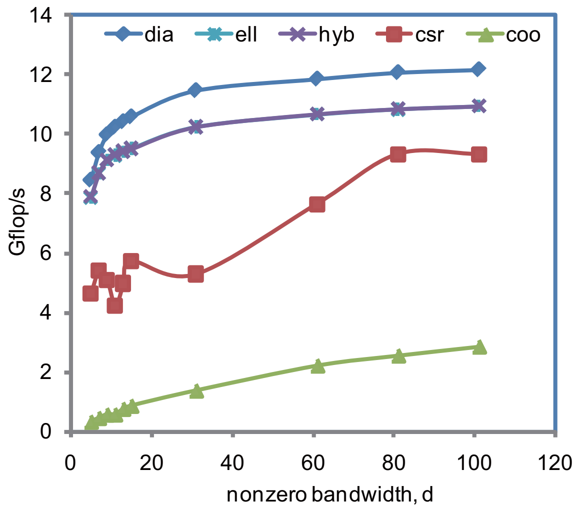

Fig. 7 Efficiency of spMV versus nonzero bandwidth of matrix, *d*, in double precision

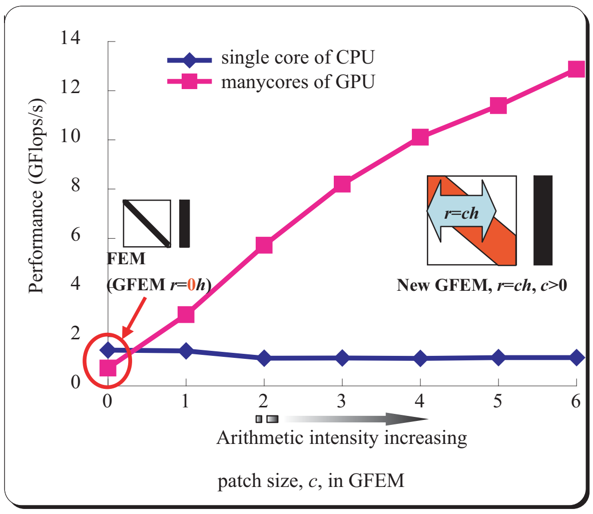

Fig. 8 Floating point performance of the new GFEM on manycores

that the performance of the FEM has no obvious difference on the single core of the CPU and the manycores of the GPU, although the GPU processor offers much higher floating point performance (peak performance: 78GFlops) than the single core CPU (peak performance: 10GFlops). This clearly shows that the high floating point performance of the GPU is underutilized if simply continuing to use the low order FEM on the manycore architecture.

Second, we increase the patch size and measure the performance achieved on the CPU and the GPU. On the GPU, when increasing the patch size *r*, i.e., reducing the sparsity of the global stiffness matrix or increasing the order of the method, the performance increases significantly. This means the method well utilizes the high floating point performance of the manycores. This implies that the new GFEM, although traditionally thought to be expensive from the classical viewpoint (in the PC or the single core times), might be reservedly hardware efficient on the manycore architecture.

On the other hand, it is observed that on the CPU increasing the patch size does *not* result in better and even worsen the performance. This exactly explains the reason why the methods leading to a dense matrix, or the high order methods, have been traditionally deemed to be *not* computing efficient in the PC or the single core times.

These tests, to a certain extent, reflect the philosophy shift in what is a “good” numerical algorithm in the near future—in the PC or the single core times, memory is taken for granted and flops are expensive, whereas in the manycore times, memory becomes the new bound while flops become free. By tuning the sparsity of the stiffness matrix, the new GFEM can release the compute power of the manycores while at the same time offering the high order of accuracy and convergence.

What observed in the above should also apply to meshfree methods because we are only focused on solving the system of linear equations. For both the new GFEM and the meshfree method, their global stiffness matrices share one common feature that the stiffness matrix becomes dense for a high accuracy of analysis while the size of the system of linear equations does not change (like that shown in Fig. 6 (a)-(d)). Since the above tests are only focused on solving the system of linear equations, it is quite reasonable to believe that the meshfree method also has the same hardware-efficiency on the manycore GPU in terms of solving the system of linear equations.

Noted is that element assembly for the meshfree/GFEM approximation is also expensive. However, as the element assembly is easy to parallelize and it is also method-dependent, the part of work leaves out of consideration in the paper. Relevant discussions on this aspect can be found in references (Cecka *et al*. 2011, Karatarakis *et al*. 2013).

9. Conclusions

Traditionally we are not prone to consider a method efficient when it leads to a moderately dense stiffness matrix. This is particularly true in the PC or the single core times. However, in this paper we tried out a new way of thinking in the context of the GFEM without extra dof. By increasing the nonzero bandwidth of the global matrix (the moderately dense matrix), the new GFEM increases accuracy and convergence (flops requirements) while keeping the same the number of dofs (memory requirements) to take advantages of the high floating point performance of the emerging manycore architecture. The result reported herein is preliminary. But it does reveal that the new GFEM well utilizes the high floating point performance of the manycore GPU by the high order of accuracy and convergence; the "expensive" method is found to be reversely hardware-efficient on the emerging manycore architecture, leading to both high accuracy and high hardware efficiency, which is a win-win strategy. This finding sends a new message to the class of "expensive" methods.

Due to the fact that the memory intensive S&E applications cannot simply benefit from the increasing number of cores, the "performance gap" has a tendency to become wider on the emerging manycore architecture. In the community of computer science, a motion being pushed forward is to *co-design* numerical algorithms with the emerging architecture (DOE Exascale Initiative Roadmap 2009, DOE Office of ASCAC 2010). The "co-design", currently a hot topic in exascale computing, should be, in author's opinion, to shorten the "performance gap" to improve energy effectiveness (Tian and Sun 2013, Tian 2012). By the piece of work, the author wishes, like what illustrated in Fig. 3, to open a possibility of the "co-design".

References

Babuška, I. and Melenk, J.M. (1997), "Partition of unity method", *Int. J. Numer. Meth. Eng.*, **40**, 727-758.

Babuška, I., Caloz, G. and Osborn, J.E. (1994), "Special finite element methods for a class of second order elliptic problems with rough coefficients", *SIAM J Numer. Anal.*, **31**, 945-981.

Cecka, C., Lew, A. and Darve, E. (2011), "Assembly of finite element methods on graphics processors", *Int. J. Numer. Meth. Eng.*, **85**(5), 640-669.

DOE Exascale Initiative Roadmap (2009), Architecture and Technology Workshop, San Diego, December.

DOE Office of Science Summary report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee (2010), The opportunities and challenges of exascale computing.

Duarte, C.A. and Oden, J.T. (1996), "An *h-p* adaptive method using clouds", *Comput. Meth. Appl. Mech. Eng.*, **139**(1-4), 237-262.

Duarte, C.A., Babuska, I. and Oden, J.T. (2000), "Generalized finite element methods for three-dimensional structural mechanics problems", *Comput. Struct.*, **77**, 215-232.

Duarte, C.A., Hamzeh, O.N., Liszka, T.J. and Tworzydlo, W.W. (2001), "A generalized finite element method for the simulation of three-dimensional dynamic crack propagation", *Comput. Meth. Appl. Mech. Eng.*, **190**, 2227-2262.

Duarte, C.A. and Kim, D.J. (2008), "Analysis and applications of a generalized finite element method with global-local enrichment functions", *Comput. Meth. Appl. Mech. Eng.*, **197**(6-8), 487-504.

Karatarakis, A., Metsis, P. and Papadrakakis, M. (2013), "GPU-Acceleration of stiffness matrix calculation and efficient initialization of EFG meshless methods", *Comput. Meth. Appl. Mech. Eng.*, **258**, 63-80.

Lancaster, P. and Salkauskas, K. (1981), "Surfaces generated by moving least squares methods", *Math. Comput.*, **37**, 141-158.

Melenk, J.M. and Babuška, I. (1996), "The partition of unity finite element method: basic theory and applications", *Comput. Meth. Appl. Mech. Eng.*, **139**, 289-314.

Oden, J.T., Duarte, C.A. and Zienkiewicz, O.C. (1998), "A new cloud-based *hp* finite element method", *Comput. Meth. Appl.Mech. Eng.*, **153**(1-2), 117-126.

O'Hara, P., Duarte, C.A. and Eason, T. (2009), "Generalized finite element analysis for three dimensional problems exhibiting sharp thermal gradients", *Comput. Meth. Appl. Mech. Eng.*, **198**, 1857-1871.

Simone, A., Duarte C.A. and Van der Giessen E. (2006), "A generalized finite element method for polycrystals with discontinuous grain boundaries", *Int. J. Numer. Meth. Eng.*, **67**, 1122-1145.

Strouboulis, T., Babuška, I. and Copps, K. (2000), "The design and analysis of the generalized finite element method", *Comput. Meth. Appl. Mech. Eng.*, **181**(1-3), 43-69.

Strouboulis, T., Copps, K. and Babuška, I. (2000), "The generalized finite element method: an example of its implementation and illustration of its performance", *Int. J. Numer. Meth. Eng.*, **47**, 1401-1417.

Strouboulis, T., Copps, K. and Babuška, I. (2001), "The generalized finite element method", *Computer Methods in Applied Mechanics and Engineering*, **190**(32-33), 4081-4193.

Strouboulis, T., Zhang, L. and Babuška, I. (2003), "Generalized finite element method using mesh-based handbooks: application to problems in domains with many voids", *Comput. Meth. Appl. Mech. Eng.*, **192**, 3109-3161.

Strouboulis, T., Zhang, L. and Babuška, I. (2004), "*p*-version of the generalized FEM using mesh-based handbooks with applications to multiscale problems", *Int. J. Numer. Meth. Eng.*, **60**, 1639-1672.

Strouboulis, T., Zhang, L., Wang, D. and Babuška, I. (2006), "A posteriori error estimation for generalized finite element methods", *Comput. Meth. Appl. Mech. Eng.*, **195**, 852-879.

Strouboulis, T., Babuška, I. and Hidajat, R. (2006), "The generalized finite element method for Helmholtz equation: theory, computation and open problems", *Comput. Meth. Appl. Mech. Eng.*, **195**, 4711-4731.

Strouboulis, T., Hidajat, R. and Babuška, I. (2008), "The generalized finite element method for Helmholtz equation, part II: effect of choice of handbook functions, error due to absorbing boundary conditions and its assessment", *Comput. Meth. Appl. Mech. Eng.*, **197**, 364-380.

Tian, R. (2012), "Co-design thinking towards exascale computing", *Inform. Tech. Letter*, **10**(3), 50-63. (in Chinese)

Tian, R. and Sun, N. (2013), "Some considerations about exascale computing in China", *Commun. China Comput. Feder.*, **9**(2), 52-60. (in Chinese)

Tian, R. (2013), "Extra-dof-free and linearly independent enrichments in GFEM", *Comput. Meth. Appl. Mech. Eng.*, to appear.

http://www.top500.org/project/linpack/

http://nvworld.ru/files/articles/calculations-on-gpu-advantages-fermi/fermipeformance.pdf

http://code.google.com/p/cusp-library/