Proc. 7th Intl Conf. on High Performance Computing and Grid in the Asia Pacific Region

Scripting High Performance Earth Systems Simulations on the SGI Altix 3700

Matt Davies, Lutz Gross, Hans Mühlhaus
Earth Systems Science Computational Centre
The University of Queensland
Brisbane, Australia
{matt,gross,muhlhaus}@esscc.uq.edu.au

Abstract

A Python language scripting interface for optimised numerical software written in C/C++ has been designed to facilitate the rapid development of 3D parallel simulations on the Altix 3700. A recent advance focusing on the extensions implemented within a high performance finite element kernel module is described. Performance issues, measurements and results on the Altix 3700 are presented.

1. Introduction

The modeling library *escript* has been developed as a module extension of the scripting language Python [9] to facilitate the rapid development of 3D parallel simulations on the Altix 3700. The finite element kernel library, *Finley*, has been specifically designed for solving large-scale problems on ccNUMA architectures and has been incorporated as a differential equation solver into *escript*. While both projects are under continuous development, the current version has been applied to 3D, anisotropic and viscoelastic convection models of the Earth's mantle. In the *escript* programming model, Python scripts orchestrate numerical algorithms that are parallelised behind-the-scenes in *escript* module calls, without low-level technical threading implementation by the *escript* user. In particular, the *Finley* module extends the *escript* functionality of data array abstraction with functions that assemble a system of linear equations from a linear boundary value problem defined over a mesh.

The paper is organised as follows: In section 2 a short overview on the SGI Altix 3700 architecture is presented. An introduction to the finite element code *Finley* and its parallelisation for ccNUMA architectures is then provided in section 3. Section 4 outlines some performance results with analysis and discussion. Finally, in section 5, simulation results for a 3D mantle convection simulation obtained with *Finley* and the SGI Altix 3700 are presented.

2. The SGI Altix 3700 Architecture

The Earth Systems Simulator (ESS) situated at the University of Queensland is a 208-processor SGI Altix 3700 cache-coherent NUMA (ccNUMA), distributed shared-memory (DSM) "supercluster". The system hardware is based on the SGI Origin S^2MP architecture (see [4]) with SGI NUMAflex interconnect technology (first used in the SGI Origin 3000 [2]), the Intel Itanium 2 processor, DDR SDRAM DIMMs and supports a variety of PCI cards and adaptors. The system software administering the system hardware is based on the SGI Advanced Linux Environment with SGI ProPack, configured to operate as one node under a single system image.

The 208 processor elements of the ESS are all Itanium 2 Madison processors, each with a 3Mb L3 write-back cache, 256Kb write-back L2 cache and 16Kb write-through L1 cache, all of which are located on-die. The ESS has 2Gb of memory local to each node board.

The NUMAflex network of the ESS is implemented in a dual-plane "fat tree" topology interconnecting the basic building-block components (or "bricks") in a modular manner as shown in figure 1. The compute brick (or "C-brick") consists of two node boards, each supporting local SDRAM memory and two Itanium 2 processors connected to an ASIC via a single front side bus. The ASIC acts as a crossbar between the local memory, processors, network and I/O interfaces and is internally connected to the opposing ASIC of the other node board in the C-Brick via a 6.4Gb/s full duplex NUMAlink 4 interconnect. The ASIC is also externally connected to a router brick (or "R-brick") by a 3.2Gb/s full duplex NUMAlink 3 interconnect. The R-brick in-turn acts as a high-speed switch, routing network packets between the C-bricks and other system components including memory bricks (or "M-brick"s), I/O expansion bricks (or "IX-brick"s), PCI expansion bricks (or "PX-brick"s), data bricks (or "D-brick2"s) and other R-bricks in the network. Refer to [2] for further information on the topology.

The global shared memory addressing of the Altix 3700 is implemented in the C-Brick ASIC, interfacing the snooping operations of the Itanium 2 processor to NUMAflex protocol which is directory-based. The ASIC maintains an internal directory containing the most re-

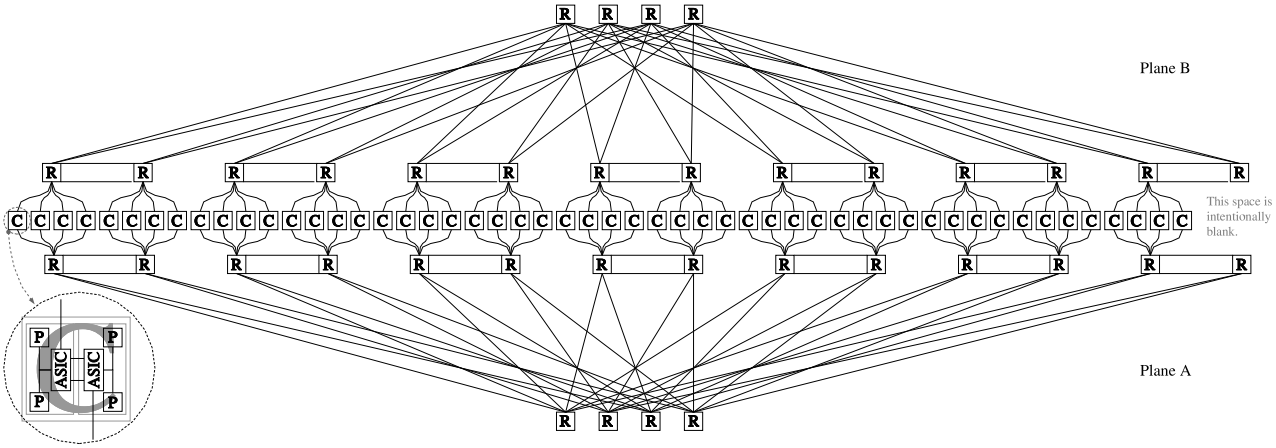

Figure 1. The Dual Plane Fat Tree Topology

cent cache coherency information for each cache-line the processor fetches. For each cache-line, a bit vector flagging which other node boards have a copy of the particular cache-line is maintained. This arrangement permits direct cache-line status transactions and updates, implemented by way of an invalidation strategy [11]. An additional 3% of local memory is reserved for inactive directory entries not found in the ASIC. Cache-line data and directory information are loaded simultaneously, reducing delays as a result of the coherency scheme [11].

3. Finley

A direct application of the finite element method is the solution of initial boundary value problems (BVPs). The *escript* Python module provides an environment to solve these problems through its core finite element library *Finley.* Within *escript,* high-level numerical algorithms including specialised nonlinear solvers and time-differencing schemes can be rapidly developed. For instance, an unsteady initial BVP can be transformed into a sequence of steady BVPs to be solved at each time step. The coefficients of the steady, linear BVP can then be provided to *Finley* to assemble a system matrix for the given domain and unstructured mesh. Although the application of *escript* is not restricted to the solution of initial BVPs, we will present the usage of *escript* and *Finley* in this context.

The steady, linear second order BVP for an unknown function u is processed by *Finley* in the following templated system of PDEs in tensorial notation:

$$-(A_{ijkl}u_{k,l})_{,j} - (B_{ijk}u_k)_{,j} + C_{ikl}u_{k,l} + D_{ik}u_k = -X_{ij,j} + Y_i. \tag{1}$$

where u_k denote the components of the solution u, and for any function Z, $Z_{,j}$ denotes the derivative of Z with respect to the jth spatial coordinate. The Einstein summation convention also applies where a duplicate subscript within a term implies summation over all possible values of that subscript. The coefficients A, B, C, D, X, and Y are functions of their location in the physical domain that must be defined at the quadrature points within each finite element.

Finley accepts a system of (natural) boundary conditions given by:

$$n_j(A_{ijkl}u_{k,l}+B_{ijk}u_k)+d_{ik}u_k = n_jX_{ij}+y_i \text{ on } \Gamma_i^N \tag{2}$$

where n denotes the outer normal field of the domain and A, B and X are as for (1). d and y are coefficients defined on the boundary, Γ. Here, $\Gamma_i^N$is a portion of the boundary where the boundary condition applies. Moreover, the Dirichlet boundary condition:

$$u_i = r_i \text{ on } \Gamma_i^D \tag{3}$$

where Γ_i^D is subset of Γ such that$\Gamma_i^D \cup \Gamma_i^N = \Gamma$, is also accepted. The right-hand side r_i is a function defined on the boundary. Within *Finley*, the sets Γ_i^D are represented through a characteristic function q defined by

$$q(x) = \begin{cases} 1 & x \in \Gamma_i^D \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

for all i. The functions r and q are defined at the nodes of the finite element mesh. In the discussion that follows the BVP defined by equations (1)-(4) will be referred to as the *Finley Boundary Value Template* or *Finley BVT*.

A special form of the *Finley BVT* for the case of single PDE for a scalar, unknown function u can be derived from (1) as:

$$-(A_{jl}u_{,l})_{,j}-(B_ju)_{,j}+C_lu_{,l}+Du = -X_{j,j}+Y \ , \tag{5}$$

In this case, the accepted natural boundary condition (2) is given in the form:

$$n_j(A_{jl}u_{,l} + B_ju) + du = n_jX_j + y \text{ on } \Gamma^N \tag{6}$$

and constraints are given by:

$$u = r \text{ on } \Gamma^D. \tag{7}$$

Finley obtains a discretisation of the *Finley BVT* from the variational formulation. A solution u is sought which fulfills the constraints (3) and

$$\int_\Omega v_{i,j}(A_{ijkl}u_{k,l} + B_{ijk}u_k) + v_i(C_{ikl}u_{k,l} + D_{ik}u_k)\,dx + \int_\Gamma v_i d_{ij} v_j\,dS = \int_\Omega v_{i,j}X_{i,j} + v_i Y_i\,dx + \int_\Gamma y_i v_i\,dS \quad (8)$$

for all trial functions v with $v_i = 0$ on Γ_i^D. The set Γ is a superset of all Γ_i^N where the functions d_{ik} and y_i are set to zero on $\Gamma - \Gamma_i^N$.

Finley uses isoparametric finite elements on unstructured meshes to discretise the variational problem (8). Available elements shapes are line, triangle, quadrilateral, tetrahedron, and hexahedron of orders one and two. The discretisation leads to a system of linear equations with a sparse coefficient system matrix. *escript* also provides a set of iterative Krylov subspace methods [10] to solve the arising linear system.

3.1. A Scripted Example

The current section demonstrates the use of *Finley* and *escript*. Assume that a solution to the diffusion equation

$$u_{,t} - u_{,jj} = f \quad (9)$$

for the scalar, time-dependent function u on the unit square is required. In this case, $u_{,t}$ is the derivative of u with respect of time and we will assume f is a given source term identical to one. At an initial time, u is known to be a Gaussian profile with its peak at $c = (\frac{1}{2}, \frac{1}{2})$ and analytical form $u(x) = \frac{1}{10}e^{-20\|x-c\|}$. The values of u at the bottom boundary $x_1 = 0$ are fixed in time.

Application of the Euler scheme

$$u^{(t-\Delta t)} = u^{(t)} - u_t^{(t)}\Delta t \quad (10)$$

with step size Δt produces a steady linear differential equation

$$u^{(t)} - \Delta t\, u_{,jj}^{(t)} = \Delta t\, f + u^{(t-\Delta t)} \quad (11)$$

at each time step t. Equating like terms against the *Finley BVT* for the single PDE (5) yields the template relations:

$$\begin{array}{rclrcl} A_{ij} & = & \Delta t\,\delta_{ij} & B_j & = & 0 \\ D & = & 1 & C_l & = & 0 \\ X & = & 0 & Y & = & \Delta t\, f + u^{(t-\Delta t)} \end{array} \quad (12)$$

As the implicit natural boundary condition can be written:

$$\frac{\partial u^{(t)}}{\partial n} = 0 \quad (13)$$

like terms are similarly equated against the *Finley BVT* for boundary conditions (6) yielding further template relations:

$$d = y = 0 \quad (14)$$

Finally, in implementation of the Dirichlet condition, the *Finley BVT* equation (7) requires a characteristic function q as defined in (4) and given by:

$$q(x_0, x_1) = \begin{cases} 1 & x_1 = 0 \\ 0 & \text{otherwise} \end{cases} \quad (15)$$

With the template parameters correctly identified, a Python script is then written solve this problem:

```
# import escript and finley
from escript import *
from Finley import *
# step size
dt=0.1
# set 20x20 mesh over [0,l0]x[0,l0]
msh=Rectangle(20,20,1,l0=1,l1=1)
# source term: constant 1.0
f=Scalar(value=1.0,where=msh.Elements())
# get the coordinates of the nodes:
x=msh.Nodes().getX()
# initial value for u:
u=1./10.*exp(-20.*length(x-[0.5,0.5]))
# start of iteration
t=dt
while t<=1.
  # assemble linear system Mu=b
  M,b=msh.assemble(A=[[dt,0],[0,dt]],\
    D=1,Y=dt*f+u,q=x[1].whereZero(),r=u)
  # solve Mu=b
  u=M.solve(b)
  # next time step
  t+=dt
```

`f=Scalar(...)` creates the source term as an *escript* data object describing a scalar value. The argument `where=msh.Elements()` associates the value to the elements of the finite element mesh, `msh`. Values associated with the elements of a finite element mesh are actually associated with the element's quadrature nodes. The value of `f` is initialised to 1.0. *escript* avoids copying this constant value to the elements and stores the value efficiently in this case. The statement `x=msh.Nodes().getX()` returns the coordinates of the nodes of the finite element mesh as an *escript* data object representing a vector value associated with the nodes. In the next statement, the initial value for u is set. *escript* automatically converts the list `[0.5,0.5]` into an *escript* data object associated with nodes in context with the subtraction from the data object `x` associated with nodes. The result is a data object representing a vector tied to nodes. The final result for `u` is a scalar object on nodes.

The method `msh.assemble(...)` calculates the stiffness matrix and right hand side vector using the finite element discretisation defined by the mesh `msh` to the left and right hand side in the variational problem (8) and the constraints (3). It is expected that the characteristic functions `q` and the values of the constraints `r` are are associated with nodes. The coefficients `A`, `D` and `Y` must

be defined on elements. In this case, `A` and `D` are defined as a list and constant value, respectively. *escript* automatically converts the given values into matrix and scalar data objects associated with elements. In a manner similar to that described for `f`, the values are not copies to the elements but stored as single values used by all elements when referring to the corresponding coefficient. The situation for `Y` is slightly more complicated. In fact the value for `Y` is defined by an expression involving data tied to elements (`f`) and data tied to nodes (`u`). In this situation, *escript* automatically invokes an interpolation of the data associated with the nodes to data associated with the elements before the operation. In this case addition is performed. Notice, that `f` is set to a constant value (=1.0). As `u` will have an individual for each element the constant value is copied to each element before the addition can be performed.

3.2. Parallelisation

Where explicit time-integration schemes are not applicable, the solution of systems of linear equations is the most computationally expensive operation within a simulation. In this case, an efficient parallelisation of the linear solver is essential to achieving improved scalability. As iterative Krylov subspace methods are based on the fundamental operations of vector update, scalar product and matrix-vector product (see [10]), they are appropriate for fine-grain parallelisation.

This section will briefly address several issues arising from the parallelisation of the *escript* BiCGStab linear solver and the *Finley* assembly routine with C/C++ and OpenMP [1]. The discussion will exclude the preconditioning of the linear system, as the intent is to present a general overview.

3.2.1. The Matrix Vector Product In each iteration step of the BiCGStab algorithm, the matrix-vector product is the most computationally expensive task. With the current implementation of *escript*, system matrices are stored in the block compressed sparse row (CSR) data structure (refer to [7] for details). The matrix-vector product with input vector `x` and output vector `y` is given as:

```
do i=1,m
  do k=ptr(i),ptr(i+1)-1
    y(i)=y(i)+entry(k)*x(index(k))
  end do
end do
```

where `m` is the number of rows of the stiffness matrix, `entry(ptr(i)...ptr(i+1)-1)` are the non-zero entries of the `i`-th row of the stiffness matrix, and `index(ptr(i)...ptr(i+1)-1)` define the corresponding column indices. The outer `i`-loop is statically parallelised with respect to workload scheduling (refer to [1]). To avoid communication overhead for the output array `y`, the Altix 3700 system software delays physical page mapping until first use (or "touch"), allowing pages to be instantiated in parallel and distributed with the thread nodes. It is critical that the physical memory pages for all other arrays (`ptr`, `x`, `entry` and `index`) are distributed in a compatible manner before entering the `i`-loop. For example, to avoid nonlocal memory access of the array `entry`, the pages containing elements `entry(ptr(i)...ptr(i+1)-1)` are instantiated on the thread node of row `i`. Note that on the computational border, cache-lines of the page containing element `entry(ptr(i))` can be shared by two processors without detriment to performance. This is due to the nonblocking cache-coherency algorithm. In the case of a cache overflow, the volume of nonlocal memory access for the matrix-vector product is at most of the size of a cache line.

The fetch operation `x(index(k))` requires additional communication if the requested elements of `x` are not stored on the thread node of row `i`. This case is minimised if non-zero entries in each row of the matrix are close to the main diagonal, or alternatively if the matrix bandwidth given by:

$$max\{|i - \mathtt{index}(k)| \\ |i = 1, \mathtt{m};\ k = \mathtt{ptr}(i),\ \mathtt{ptr}(i+1) - 1\}$$

is small. The Cuthill-McKee algorithm [3] is a heuristic that can be used with unstructured meshes to obtain a near-optimal ordering. For each row `i`, the indices `index(ptr(i))`, ..., `index(ptr(i+1)-1)` are stored in order of increasing value so that all entries read into cache can be reused before being overwritten.

3.2.2. The Stiffness Matrix Assembly The stiffness matrix and the right hand side of the linear system are assembled from element matrices that arise from a discretisation of the variational problem (8) for each mesh element. An element matrix is added onto the stiffness matrix and the right hand side vector as an overlay where, for each entry of the element matrix, the row and column indices are given by the underlying mesh node ordering. While the calculation of the element matrices is parallel by nature, adding them to the stiffness matrix and the right hand side vector as an overlay can result in memory contention. To overcome this problem, *Finley* users a colouring algorithm whereby neighbouring elements are assigned different colours so that elements with the same colour can be added in parallel. Barrier synchronization must be applied before processing elements with the next colour.

The addition of the element matrix `em` of a given element onto the stiffness matrix is given by:

```
do i=1,N
  do j=1,N
    k=add(node(i),node(j))
    entry(k)=entry(k)+em(i,j)
  end do
end do
```

where `N` is the number of nodes describing the element and `node` is the array of the node identification numbers. The function `add(r,c)` returns the address of an entry in row `r` and column `c` as an index between `ptr(r)`

and `ptr(r+1)-1` where `index(k)=c`. If the element is assigned to a node different from the thread node which stores row `node(i)` of the stiffness matrix, the cache line of `entry(k)` must be fetched to the element processor to perform the update. While a colour is being processed, no other element can update the cache line except in the bordering case where it is shared by two rows. The loaded cache line will persist for a given colour if cache overflow is avoided.

Where nonlocal memory access cannot be avoided, a cache-line should be fetched from node as topologically close as possible for maximum efficiency. For this reason, elements are ordered by the increasing mean value of their node ordering. Together with node bandwidth minimising algorithms, an element is as close as possible to the nodes where the rows it contributes to are stored.

The assembly process presented here assumes that the sparsity pattern of the stiffness matrix is known beforehand. Alternatively, this requires that the values of the entries of `ptr` and `index` are known before actual nonzero values are calculated. To derive the matrix pattern, a colouring algorithm similar to the one used in the assembly process is used.

4. Finley and SGI Altix 3700 Performance

The performance of the Finley parallelisation and the Altix 3700 has been studied with abstract scalar and vector test cases. The abstract test cases are solved using *escript* and *Finley* with 1 to 128 processors.

The abstract scalar test case studied in this paper is the elliptic BVP

$$u - u_{,jj} = f \tag{16}$$

on the n-dimensional unit square with Dirichlet boundary conditions on the entire boundary. f is selected such that the solution of the problem is

$$u = \sum_{i=1}^{n} x_i^2 \tag{17}$$

In comparison, the abstract vector test case studied in this paper is the elliptic BVP

$$-(u_{k,k}\delta_{ij} + u_{i,j} + u_{j,i})_{,j} = f_i \tag{18}$$

on the n-dimensional unit square with Dirichlet boundary conditions on the entire boundary. f is selected such that the solution of this problem is

$$u_j = \sum_{i=1, i\neq j}^{n} x_i^2,\ j = 1...n \tag{19}$$

4.1. Discussion

The preconditioned BiCGStab algorithm requires 2 matrix-vector products, 4 inner products, and 2 preconditioner solves per iteration. A Jacobi preconditioner has been used to simplify the analysis, resulting in an algorithm of 2 matrix-vector products and 6 inner products. The number of iteration steps needed to reach a given tolerance depends on the size of the assembled system matrix. Consequently, the complexity of solving a linear system grows super-linearly where the complexity per iteration step is linearly dependent on the number of unknowns. For purposes of comparison between problems of different sizes, the discussion is restricted to the computation of a single iteration of the BiCGStab algorithm.

Timings were recorded for the solution of the scalar and vector test cases, (16) and (18), in 2D and 3D for a distribution of thread numbers and mesh resolutions on the SGI Altix 3700. Typical test results generated with *escript* and *Finley* for the scalar test case in 3D are shown in figure 2. In this case, the number of elements used was

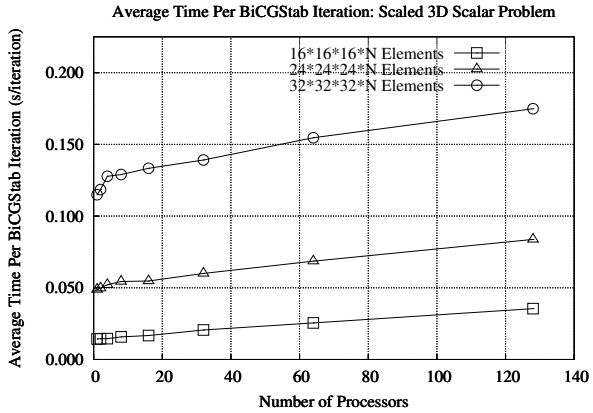

Figure 2. Sample 3D results.

scaled to maintain a constant problem size per thread. The effects of the NUMA architecture are evident with the increase of compute time between 4 and 8 nodes caused by nonlocal memory access during the two matrix-vector products. As described in section 2, the four processors within a C-brick are connected by a 6.4Gb/s full duplex NUMAlink 4 interconnect. Outside of the C-brick, each node board is connect to a R-brick by only a single 3.2Gb/s NUMAlink 3 interconnect. For the case of two threads, the memory access is local. For the case of 4 threads, all nonlocal memory is accessed via a 6.4Gb full duplex channel. In both cases, the nonlocal memory access for the matrix-vector product is achieved at a higher bandwidth. In comparison, for the case of 8 threads, nonlocal memory is accessed with 4 hops in the worst case (one 6.4Gb/s channel and three 3.2Gb/s channels). The 3.2 Gb/s channel is the limiting bandwidth for nonlocal memory access with more than 4 threads.

A simple model is proposed to predict the the execution time for a BiCGStab iteration with m unknowns on N processors:

$$T(m, N) = T_0 + T_1\frac{m}{N} + T_2 m^{(n-1)/n} + T_3 f(N) \tag{20}$$

Here $T_0 + T_1 \frac{m}{N}$ represents the computational cost. The communication cost of the matrix-vector product is proportional to the bandwidth of the system matrix. The bandwidth can be estimated by the number of mesh points in a cross-section of the domain. As $\sqrt[n]{m}$ is the number of mesh nodes in each spatial direction, the number of nodes of a cross-section is estimated by $m^{(n-1)/n}$ where n is the spatial dimension. It follows that the term $T_2 m^{(n-1)/n}$ represents the cost of nonlocal memory access. The last term, $T_3 f(N)$, represents the time spent in global communication operations such as barrier synchronizations and the reduction of partial results of scalar products across threads.

A logarithmic growth in the global communication cost was expected (i.e. $f(N) = O(log(N))$). However, the timings obtained for a small number of elements suggested that the global communication cost is linear (i.e. $f(N) = O(N)$). A least squares fit was applied to estimate the coefficients of (20) for each of the cases of logarithmic and linear growth in the global communication cost. For each test case, a variety of time measurements covering the parameter space for N ($1 \leq N \leq 128$) and m ($5 \times 10^3 \leq m \leq 10^7$) were obtained. It is assumed that $T_0 = 0$. Table 1 shows the fitted values of the coefficients. The 6th column is the correlation coefficient and the 7th column is the defect (the sum of the square of the differences from the model). A small defect infers a good correlation between the data and the model. In this study, both the correlation coefficient and the defect indicate that linear growth is a more appropriate model for the global communication cost.[1] While it is expected that the fit coefficient T_3 is independent of the linear system, T_3 was fitted with a value of approximately 50 for the case $f(N) = O(N)$ and values between 50 and 500 for the case $f(N) = O(log(N))$. In comparison with the fit for the former case, the degree of variability in the later case indicates that a linear model for global communication is likely.

The majority of the communication cost of the matrix-vector product is determined by the bandwidth of the matrix and by the number of entries per matrix row. The estimated values of T_2 indicate that the communication cost is similar for 2D and 3D problems. It is noted that the values for the vector case are slightly larger due to the increased number of entries per row.

As the mesh is both regular and structured and the finite elements are second order, the average number of nonzero entries per row is 16 for the 2D scalar test case, 32 for the 2D vector test case, 59 for the 3D scalar test case, and 177 for the 3D vector test case. As the BiCGStab algorithm requires 24 floating point operations per unknown per iteration step, each iteration step requires 40, 56, 83 and 201 floating point operations per unknown for each test case respectively. Therefore, an increase of the fit coefficient $T_1$by the factors 56/40=1.4, 83/56=1.48 and 201/56=2.42 is expected between corresponding 2D scalar and 2D vector test cases, corresponding 2D vector and 3D scalar test cases, and corresponding 3D scalar and 3D vector test cases respectively. These predicted ratios compare with the observed ratios of 1.4, 1.5 and 2.5.

1 At the time of submission, it could not be confirmed with SGI if the complexity of the algorithm for global communication is linear with the number of threads.

Case	$f(N)$	T_1	T_2	T_3	Corr.	Def.
Sca2D	N	0.379	4.27	41.9	0.994	100
	log(N)	0.368	5.95	114	0.989	106
Vec2D	N	0.549	5.53	36.7	0.998	173
	log(N)	0.542	6.93	51.2	0.997	258
Sca3D	N	0.826	0.945	33.2	0.998	245
	log(N)	0.822	0.980	306	0.997	281
Vec3D	N	2.04	1.02	59.1	1.00	423
	log(N)	2.03	1.08	580	0.99	513

Table 1. Timing model fitted coefficients.

5. A Case Study: Mantle Convection

escript is designed to facilitate the development of 3D parallel Earth systems simulations on the Altix 3700 for a variety of users including geophysicists, mathematicians and engineers. With the kernel finite element library *Finley*, *escript* has been applied to 3D nonlinear models arising from the study of the convection of the Earth's mantle. Such models represent the rock as an incompressible, highly viscous material. The model discussed here is based on 3D anisotropic non-Newtonian mantle convection.

The governing equations of mantle convection [8, 5] consist of the equations of motion of the material points of the continuum and the heat equation. These equations are given by:

$$\sigma'_{ij,j} - p^{th}_{,i} + Ra^{(c)}\, T\, g_i = 0$$

$$T_{,t} + v_j\, T_{,j} = T_{,jj} + \frac{Di^{(c)}}{Ra^{(c)}}\, \sigma'_{ij}\, D_{ij}$$

The equations of motion for this study are subsequently obtained by substitution of the anisotropic constitutive relationships [6], relating the velocity gradients to the components of the stress tensor into the stress equilibrium equations. In this case, the stress is given by:

$$\sigma_{ij} = 2\,\eta\, D'_{ij} + 2\,(\eta_S - \eta)\,\Lambda_{ijkl}\, D'_{kl} - p\,\delta_{ij}$$

where

$$\Lambda_{ijkl} = \frac{1}{2}\,(n_i\, n_k\, \delta_{lj} + n_j\, n_k\, \delta_{il} + n_i\, n_l\, \delta_{kj} + n_j\, n_l\, \delta_{ik}) - 2\, n_i\, n_j\, n_k\, n_l$$

Here η and η_S are the normal and the shear viscosities respectively. The material behavior is isotropic if $\eta = \eta_S$. The tensor $\Lambda_{ijkl} D'_{kl}$ maps the deviatoric component of the stretching D_{kl} with respect to the global coordinates

onto the shear strain rate with respect to the principal axes of anisotropy (see [6] for details). As shown above, the components of the Λ_{ijkl} tensor can be expressed in terms of the components of the normal vector (also referred to as the director) of the anisotropy surfaces. The director components transform like material surface normals and a separate evolution equation for the director is part is appended to the governing equations. The equations of motion are coupled through the temperature dependence of the density (Boussinesque approximation [8, 5]) and the temperature advection term in the heat equation. As the heat equation is advection dominated, an appropriate upwinding strategy has to be applied in the formulation of the numerical model. The equations of motion and the heat equation are nondimensionalised in the usual manner for mantle convection [8, 5]. The controlling nondimensionalised number is the *Rayleigh Number* (Ra) ranging between 10^4 and 10^9. A standard SUPG method [12] is used for both the heat equation and the director evolution equation. For the time integration of the heat equation a standard backward Euler scheme is used. The coefficient tensors of the *Finley BVT* are related to the terms of the governing equations as follows:

$$-(A_{ijkl}v^{(t)}_{k,l})_{,j} = Y_i - X_{ij,j}$$

where

$$\begin{aligned} A_{ijkl} &= \delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk} \\ &\quad +(Penalty - \frac{2}{3})\delta_{ij}\delta_{kl} + \left(\frac{\eta_S}{\eta} - 1\right)\Lambda^{(t)}_{ijkl} \\ X_{ij} &= p\delta_{ij} \\ Y_i &= RaT^{(t)}\delta_{ni} \end{aligned}$$

Similarly, the discretisation of the temperature equation is given by:

$$\begin{aligned} -(A_{ij}T^{(t+\Delta t)}_{,i})_{,j} - (B_jT^{(t+\Delta t)})_{,j} & \\ +C_jT^{(t+\Delta t)}_{,j} + DT^{(t+\Delta t)} &= Y - X_{j,j} \end{aligned}$$

where

$$\begin{aligned} A_{ij} &= \kappa\delta_{ij} + \tfrac{h}{2\sqrt{v_lv_l}}v_jC_i & D &= \tfrac{\rho c_p}{\Delta t} \\ B_j &= \tfrac{h}{2\sqrt{v_lv_l}}v_jD & Y &= DT^{(t)} \\ C_j &= \rho c_p v_j & X_j &= \tfrac{h}{2\sqrt{v_lv_l}}v_jDT^{(t)} \end{aligned}$$

Implementing the discretisation as described in section 3.1, the simulation is performed using 243,000 2nd-order finite elements with $Ra = 10^6$. In this case, a smooth, rigid container of dimensions $3 \times 3 \times 1$ is filled with a transversely isotropic, viscous fluid with the anisotropic axis initially in the direction of the x_3 coordinate. The container is heated from below with temperatures at the top and the bottom kept constant. An initial perturbation of the temperature field of the form

$$T = 1 - x_3 + \frac{1}{10}\left(cos(\pi x_1)\, cos(\pi x_2)\, sin(\pi x_3)\right)$$

instigates the convection. The average mechanical power peaks at approximately 1300 before decreasing and then finally settles into a steady-state. Figure 3 shows a visualisation of the temperature isotherms and velocity streamlines just after the peak of the average mechanical power (which is proportional to the so called Nusselt number [6]).

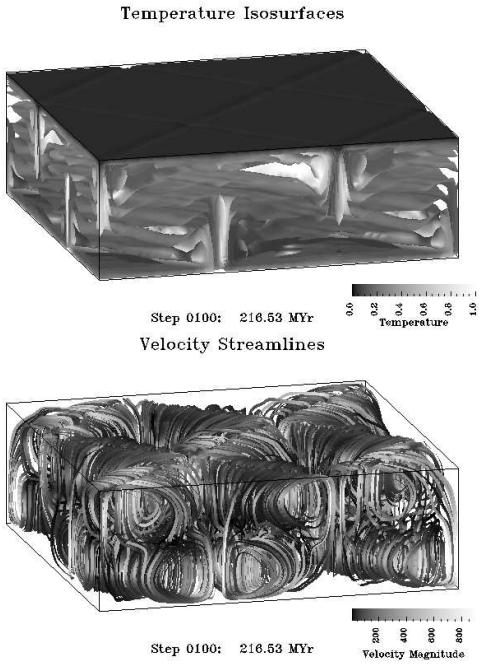

Figure 3. Mantle convection results after 216 million years. Temperature and velocity are nondimensionalised.

6. Conclusion

escript, a Python language scripting interface to optimised parallel numerical software written in C/C++ has been presented. In particular, a high performance finite element kernel module , *Finley*, has been discussed as an extension to *escript. Finley* facilitates the assembly of finite element system matrices for the solution of linear BVPs defined on unstructured meshes.

The parallel performance of the Altix 3700 for abstract test cases of variable size and threading have been investigated. A notable issue was the effect of the NUMA architecture on efficiency and scalability for larger-scale problems, indicating that fine-grain algorithms suffered performance degradation as a result of increased communication costs between C-bricks. A simple model was proposed indicating that global communication for the multi-threaded BiCGStab iteration is not scalable.

A case study focusing on the 3D numerical simulation of anisotropic mantle convection processes has been presented. Such large-scale problems in computational earth

systems simulations are the motivation for this work. Future work will further develop *escript*'s ability to undertake such problems and improve its parallel efficiency for the SGI Altix 3700.

Acknowledgment This work has been funded by the Australian Computational Earth System Simulator Major National Research Facility (ACcESS MNRF).

References

[1] OpenMP Architecture Review Board. *OpenMP C and C++ Application Program Interface (Version 2.0)*, 2002.

[2] Dick Brownell. *SGI Altix 3000 User's Guide*, 2003.

[3] Elizabeth Cuthill and J McKee. Reducing the bandwidth of sparse symmetric matrices. In *ACM National Conference*, New York, 1969.

[4] James Laudon and Daniel Lenoski. The SGI Origin: A ccNUMA highly scalable server. In *Proceedings of the 24th International Symposium on Computer Architecture (ISCA'97)*, pages 241–251, Denver, Colorado, 1997.

[5] L. Moresi, F. Dufour, and H. B. Mühlhaus. A Lagrangian integration point finite element method for large deformation modeling of viscoelastic geomaterials. *Journal of Computational Physics*, (184):476–497, 2003.

[6] H. B. Mühlhaus, F. Dufour, L. Moresi, and B. Hobbs. A director theory for visco-elastic folding instabilities in multilayered rock. *International Journal of Solids and Structures*, (39):3675–3691, 2002.

[7] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Second edition, 2000.

[8] Donald L. Turcotte and Gerald Schubert. *Geodynamics*. Cambridge University Press, second edition, 2002.

[9] Guido van Rossum. *Python Reference Manual (Version 2.3.3)*. PythonLabs, 2003.

[10] Rudiger Weiss. *Parameter-Free Iterative Linear Solvers*, volume 97 of *Mathematical Research*. Akademie Verlag, Berlin, 1996.

[11] Michael Woodacre, Derek Robb, Dean Roe, and Karl Feind. The SGI altix 3000 global shared-memory architecture. 2003.

[12] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method: Fluid Dynamics*, volume 3. Butterworth-Heinemann, Oxford, fifth edition, 2000.

