Specification of Adaptive Stencil PRK

Rob F. Van der Wijngaart

1 Background

Two important distinct types of workload transitions requiring dynamic load balancing for efficiency are:

1. An evolving mismatch between distributed data structures, exemplified by the PIC PRK; the total work (number of particles) in the system stays the same, but the dependence of parts of one data structure on parts of another data structure changes over time. The difficulty here is that any static decomposition of the data to distribute the work (in the case of PIC each worker would get a fixed part of the grid and a fixed subset of the particles) will lead to a lot of communication to satisfy the data dependences. This pattern features a continual source of load imbalance that is constant in time, and whose intensity can be specified.
2. Computational work appears and disappears intermittently, which necessitates redistribution of work. The difficulty in this case is typically that the new work depends on a specific part of a distributed data structure, so distributing it evenly among the system resources would be very inefficient with respect to communication. If the work is grid-based, a very fine distribution would also negatively affect surface-to-volume ratios. This pattern features abrupt sources of load imbalance, whose intensity and frequency can be specified.

We assume there is always enough concurrence to keep all computational resources busy, in principle. The two types often occur simultaneously in the same application. For the second type we propose a kernel inspired by Adaptive Mesh Refinement problems. It is cast as an extension of the Stencil PRK.

2 Specification

We define a background grid of a certain size, and start/stop work on smaller grids (*refinements*) with a specified frequency. A refinement does not necessarily have fewer points than the background grid, as it can have a much finer mesh spacing. We apply the same stencil operation $S(R)$, implementing the discrete divergence, to all *interior* grid points. Refinements are aligned with the background grid and are totally contained within it. At any point in time there is at most one refinement active. Each point of the background grid that falls topologically within a refinement coincides with a grid point of that refinement.

Problem parameters:
- T: total number of iterations (background grid)
- R: radius of difference stencil
- n: linear dimension of square background grid (n^2 points). The mesh spacing is one.
- r: refinement level (mesh size of refined grid is 2^{-r})
- k: linear dimension of refinement in terms of background grid cells ($k*2^r+1)^2$ refinement points)
- P: duration in terms of iterations on the background grid of one full cycle of activation of one refinement until that of the next (*period*)
- D: duration in terms of iterations on the background grid of activity on each refinement ; $D \leq P$
- d: number of iterations on a refinement per iteration on the background grid

2.1 Initialization

The input field IN_{bg} on the background grid is initialized with function $U(x, y)$, which is linear in the coordinates: $U(x, y) = c_x x + c_y y$, with $c_x, c_y >= 0$. The bottom left corner of the background grid coincides with the origin of the coordinate system. A set of four distinct refinement grids is created at the start of the program. Their input fields are initialized to all zeroes: $IN_i \equiv 0, i \in \{0, 1, 2, 3\}$.

The output fields on background and refinements are all initialized to all zeroes: $OUT_{bg} \equiv OUT_i \equiv 0, i \in \{0, 1, 2, 3\}$.

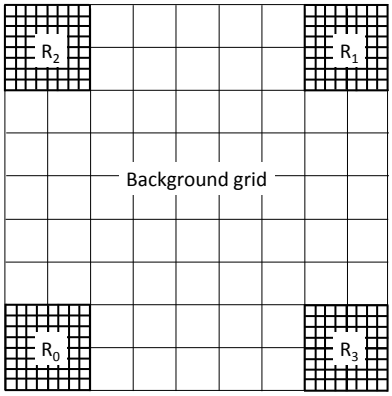
2.2 Computations

The result of each application of the stencil operation on the background grid (an *iteration* or *time step*) is accumulated in OUT_{bg}, i.e. $OUT_{bg} = OUT_{bg} + S(R)IN_{bg}$, after which the input field IN_{bg} is uniformly incremented by one. When a refinement becomes active, its input values are interpolated from the input values on the background grid at that time, using a bi-linear interpolation function ϕ: $IN_i = \phi(IN_{bg})$, evaluated on the half-open background grid cell $[i, i+1) \times [j, j+1)$ that contains the target refinement point. Subsequently, for each of the D iterations on the background grid, d stencil iterations are carried out on the refinement that was just activated, the results of which are accumulated in the corresponding output field, i.e. $OUT_i = OUT_i + S(R)IN_i$. After each iteration on the refinement, its input field IN_i is uniformly incremented by one.

2.3 Refinements

We define a set of four refinements, located at the bottom left, top right, top left, and bottom right corner of the background grid, respectively, that are activated and deactivated cyclically in that order. The bottom left corners of the refinements coincide with the following points of the background grid, respectively: $\{(0, 0), (n-1-k, n-1-k), (0, n-1-k), (n-1-k, 0)\}$. The iteration on the background grid at which refinement i ($i \in \{0, 1, 2, 3\}$) is activated for the

a^{th} time is: $P(i + 4a), a = 0, 1, 2, \cdots$. In the figure below we show an example configuration with grid and refinement parameters $n = 10$, $r = 2$, and $k = 2$. The background grid has 100 points, and each refinement has 81 points.



Example background grid with successive refinements R_0–R_3.

2.4 Verification

After all iterations have finished we compute the L^1 norm of the computed divergence over the interior points of the background grid and all refinements, normalized by the respective numbers of grid points. These values are compared to the analytical values, see Section 2.6. The absolute values of their differences are not allowed to exceed a fixed error tolerance.

To determine the L^1 norm for refinement i we need to know how many times τ it has been visited. We find: $\tau_i = d*\{\lfloor T/(P*4)\rfloor * D + \min(\max[0, T \bmod (P*4) - i*P], D)\}$. For the background grid τ_{bg} simply equals T.

In addition, after all iterations have finished we compute the L^1 norm of the prescribed input fields over all points of the background grid and all refinements, normalized by the respective numbers of grid points. These values are compared to the analytical values, see Section 2.6. The absolute values of their differences are not allowed to exceed a fixed error tolerance. To determine the L^1 norm for refinement i we need to know:

- The time step t_i (in terms of iterations over the *background* grid) when refinement i was last activated; this is because the input field of the refinement is interpolated from the input field of the background grid at that time, and its prior history is destroyed.
- The number of updates u_i of refinement i since the last interpolation.

– The initial input field value v_i on the background grid at the location that coincides with a reference point on refinement i. For convenience we pick as reference point the point of the refinement with the smallest coordinates (bottom left point), see section 2.3.

2.5 Performance metric

Performance of this kernel is reported in terms of nominal floating point operations related to the application of the stencil, to the update of the input fields, and to the interpolation, divided by the time it took to carry out all computations. If the spacing of the refinements equals that of the background grid, no floating point operations will be assigned to the interpolations, since these can be implemented efficiently and conveniently as copy operations. In all other cases we assign a fixed number of floating point operations to each refinement grid point to carry out the interpolation, corresponding to the number of operations necessary to determine the interpolant at a refinement point whose two coordinates do not coincide with any of the coordinates of the background grid.

2.6 Appendix A: Verification arithmetic

Only the linear terms of the input field make a contribution to the divergence. The linear parts of the input field on the background grid and the refinements do not change. Each iteration on a grid contributes $c_x + c_y$ to the divergence at each point. Consequently, the verification value of the computed divergence on each grid equals $\tau * (c_x + c_y)$.

To compute the normalized checksum C_i of the final input field on refinement i, we define: $f_i = (4\lfloor T/(4P) \rfloor + i)P$. With this we find:

– $\{v_i | i = 0, 3\} = \{0, (n-1-k) * (c_x + c_y), (n-1-k) * c_y, (n-1-k) * c_x\}$.

– $t_i = \begin{cases} f_i & \text{if } f_i \leq T \\ f_i - 4P & \text{if } f_i > T \end{cases}$

– $u_i = \begin{cases} \min(\max(0, \bmod(T, 4P) - i * P), D) * d & \text{if } f_i \leq T \\ D * d & \text{if } f_i > T \end{cases}$

– $C_i = v_i + (c_x + c_y) * k/2 + t_i + u_i$.

– $C_{bg} = (c_x + c_y) * (n-1)/2 + T$.

2.7 Appendix B: Clarification

As is the case for all PRK, we make sure that all required work is done, and that any work that is skipped or incorrectly carried out results in a failed verification test. Hence, each operation on the background grid and any of the refinements needs to have an effect on the final result that can be verified. We therefore accumulate each computed result into the output fields, rather than overwriting the output fields.

In a real stencil application input fields change with each iteration, requiring communications to exchange ghost point values at each iteration. To make such communication necessary in this kernel we also change the input field values after

each iteration. The simple additon of a constant throughout has no influence on the computed divergence, which simplifies the verification test.

In order to keep the amount of state that needs to be preserved during the execution of the kernel constant, we opt for a fixed set of refinements that are statically determined, but that are activated and deactivated periodically. This allows us to accumulate results on refinements into a small, fixed set of arrays.

3 Appendix C: Sample scenarios

We describe two different configurations. The total number of iterations on the background grid is considered sufficiently large that many cycles of refinements will occur, but is otherwise left unspecified. The size of the background grid to be used is the same for both scenarios ($n = 1000$).

1. $r = 1$, $k = 100$, $P = 3$, $D = 1$, $d = 1$. Refinements appear and disappear in rapid succesion, leaving little time for an automatic load balancer to respond to the variations in the load. Each refinement accounts for approximately 1% of the work on the background grid. With a modest number of workers, say 10, that means that if each refinement is assigned to the worker who is responsible for the part of the background grid that coincides with the refinement, that worker will have approximately 10% more work than the workers who do not have responsibility for any refinement. This results in a load imbalance that may be deemed acceptable. However, for a larger number of workers, say 100, the same strategy would lead to an increase in work for busy workers of 100% during times of refinement.
2. $r = 4$, $k = 6$, $P = 30$, $D = 10$, $d = 5$. Refinements contain about the same number of grid points as in configuration 1, but are now limited to a much smaller fraction of the background grid. They are activated at a rate an order of magnitude lower than in that case, and remain active 50 times longer, allowing an automatic load balancing scheme more opportunity to respond effectively to changes in the load.