Department of Computer Science, Rice University

Eureka-Style Computations in CnC

`Jacobi2D` (Iterative Convergence, Stencil Computation)

Nick Vrvilo, Shams Imam

September 15, 2014

1 Introduction

A wide range of problems, such as combinatorial optimization, constraint satisfaction problem, image matching, genetic sequence similarity, iterative optimization methods, etc., can be reduced to tree or graph exploration problems [3, 1, 4]. A pattern common in such algorithms to solve these problems is a *eureka* event, a point in the program which announces that a result has been found. Such an event usually curtails computation time by avoiding further exploration of the solution space or by causing the termination of the computation. If no eureka event occurs, the "eureka-style" computation completes when all operations in the algorithm have completed. Eureka-Style Computations (EuSCs) include search and optimization problems that could benefit greatly from speculative parallelism. The focus of this work is to implement a parallel EuSC, namely `Jacobi2D`, in CnC.

2 Convergence Iterations and Determinism

Iterative methods refer to a wide range of techniques that use successive approximations to obtain more accurate solutions to a linear system at each step [5]. Examples of iterative methods include the Jacobi method, Gauss-Seidel method and the Successive over-relaxation method. An iterative method is called convergent if the corresponding sequence converges for given initial approximations. Speculatively parallelizing an iterative algorithm results in creating tasks for computations of *future* iterations. The Jacobi2D stencil computation [6] is used to add support for the iterative convergence eureka to CnC. Inherently such computations are non-deterministic, but we can make convergent eureka computations deterministic by using Branch and Bound (BnB) algorithms.

BnB is a widely used tool for solving large scale NP-hard combinatorial optimization problems [2]. A BnB algorithm searches the complete space of solutions for a given problem for the best solution. Subproblems are derived from the originally given problem through the addition of new constraints. An objective function computes the lower/upper bounds for each subproblem. The upper bound is the worst value for the potential optimal solution, the lower bound is the best value. The entire tree maintains a global upper bound (GUB): this is the best upper bound of all nodes. Nodes with a lower bound higher than the GUB are eliminated from the tree because branching these sub-problems will not lead to the optimal solution. In many practical cases, the amount of pruning that occurs in this type of BnB algorithm can be immense.

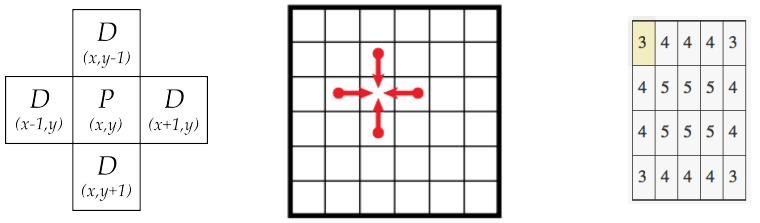
3 JACOBI2D CnC Application

`Jacobi2D` is an iterative 5-point two dimensional stencil computation (Figure 1a). The update function computes the arithmetic mean of a cell's four neighbors (top, left, bottom, and right). In this case we set off with an initial solution of 0. The left and right boundary are fixed at 1, while the upper and lower boundaries are set to 0. After a sufficient number of iterations, the system converges. Listing 1 shows the algorithm we use.

Listing 1: Jacobi2D Stencil Computation.

```
D = ... // initial input

var converged = false
while (!converged) {

  // stencil computation
  for (x in 1 to N) {
    for (y in 1 to N) {
      P(x, y) = 0.25 * (D(x, y-1) + D(x-1, y) + D(x, y+1) + D(x+1, y))
  } }

  // convergence check
  delta = 0
  for (x in 1 to N) {
    for (y in 1 to N) {
      delta = max(delta, P(x, y)
  } }
  if (delta < epsilon) {
    converged = true
  }

  D = P
}
```

Figure 1b shows the data dependencies for individual cells and these can be used to create the kernel of an iteration in CnC. Using the data dependencies it is easy to come up with the get counts for each cell (shown in Figure 1c). A CnC application of `Jacobi2D` contains of (a) a step to launch the body of an individual iteration, (b) a step to compute individual (or a chunk) of stencils, and (c) a step to perform the convergence check and determine when the application can terminate. Previous CnC implementations of `Jacobi2D` use one of two strategies:



(a)5-point 2D stencil. (b)Data dependences. (c)Get counts.

Figure 1: Data dependences and get counts for 5-point stencil.

- **Approach-1:** avoid the convergence check and instead run the algorithm for a fixed number of iterations to provide the determinism guarantee, or
- **Approach-2:** perform the convergence at the end of each iteration before spawning the steps to perform the stencil computation of the next iteration.

Approach-1 is limiting as it cannot guarantee that convergence will be reached in a pre-specified number of iterations. Approach-2 is limiting as there is now an implicit barrier at the end of each iteration that limits the speculative parallelism in the application by preventing steps from the future iterations to start.

We believe the convergence check is an orthogonal concern and should not interfere with the main parallelism in the application – computing the stencils in each iteration. Our approach is to allow future iterations to start computing the stencils as soon as their dependences are satisfied. The convergence check runs independently when its data become available and can trigger the eureka event that signals the computation to terminate. The BnB strategy is used to terminate (cancel) steps from future iterations only and allow in-flight steps from previous iterations to continue executing and also triggering eurekas. This approach allows us to detect the earliest iteration that triggered a eureka and thus produce a deterministic value. Note: Due to time constraints, our current implementation does have a maximum iteration limit, but such a limit is not actually required as look-ahead iterations can be progressively spawned, the eureka will ensure we terminate all redundant steps.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *Journal of molecular biology*, 215(3):403–410, October 1990.

[2] Jens Clausen. Branch and Bound Algorithms - Principles and Examples. *Parallel Computing in Optimization*, pages 239–267, 1997.

[3] Wei-Ming Lin, Wei Xie, and Bo Yang. Performance Analysis for Parallel Solutions to Generic Search Problems. In *Proceedings of the 1997 ACM Symposium on Applied Computing*, SAC '97, pages 422–430. ACM, 1997.

[4] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the United States of America*, 85(8):2444–2448, April 1988.

[5] Wikipedia. Iterative method. http://en.wikipedia.org/wiki/Iterative_method, 2014.

[6] Wikipedia. Stencil code: 2D Jacobi iteration. http://en.wikipedia.org/wiki/Stencil_code#Example:_2D_Jacobi_iteration, 2014.