Geometric Tiling for Reducing Power Consumption in Structured Matrix Operations *

G. Chen, L. Xue, J. Kim
Pennsylvania State University
{guilchen, lxue, jskim}@cse.psu.edu

K. Sobti, L. Deng
Arizona State University
{Kunwaldeep.Sobti, Lanping.Deng}@asu.edu

X. Sun, N. Pitsianis
Duke University
{xiaobai, nikos}@cs.duke.edu

C. Chakrabarti
Arizona State University
chaitali@asu.edu

M. Kandemir, N. Vijaykrishnan
Pennsylvania State University
{kandemir, vijay}@cse.psu.edu

ABSTRACT

This work focuses on reducing power consumption while maintaining the efficiency and accuracy of matrix computations using both algorithmic and architectural means. We transform the algorithms, in adaptation to application specifics, to translate the matrix structures into power saving potential via geometric tiling. Instead of using blind tiling, we index and partition matrix elements according to the underlying geometry to claim a better estimate and control of numerical range within and across geometric tiles, which can then be exploited for power saving.

I. ADAPTATION WITH GEOMETRIC TILING

We introduce in this section an algorithm transformation oriented towards reducing power consumption without compromising the numerical accuracy of the application. We shall describe the class of matrix-vector multiplication we are concerned with. Let $\mathcal{T}$ and $\mathcal{S}$ be two finite sets of sample points in R^3. We refer to $t \in \mathcal{T}$ as a target (point), to $s \in \mathcal{S}$ as a source. We are provided with an interaction kernel function $\alpha(t,s)$ defined on $\mathcal{T} \times \mathcal{S}$ and a function $\beta(s)$ defined on the source set $\mathcal{S}$. We are interested in evaluating another function $\gamma(t)$ defined on the target set $\mathcal{T}$ as follows:

$$\gamma(t) = \sum_{s \in \mathcal{S}} \alpha(t,s)\,\beta(s), \quad t \in \mathcal{T} \tag{1}$$

The function $\gamma(t)$ describes the response at every single target point to the function $\beta(s)$ at all the source points in $\mathcal{S}$. In matrix expression, the computation in (1) can be described as a matrix-vector multiplication:

$$\mathbf{g} = \mathbf{A}\mathbf{v}, \ \text{ with } \mathbf{A} = [\alpha(t,s)], \ \mathbf{g} = [\gamma(t)], \ \mathbf{v} = [\beta(s)] \tag{2}$$

The matrix $\mathbf{A}$ has m rows and n columns, with $m = |\mathcal{T}|$ and $n = |\mathcal{S}|$. While $\alpha(t,s)$ is the interaction between two points t and s, $\mathbf{A}$ captures the interaction between the sets $\mathcal{T}$ and $\mathcal{S}$. In the same way, any sub-matrix of A is associated with a subset of $\mathcal{T}$ and a subset of $\mathcal{S}$ and represents the interaction between the two subsets. Such geometric indexing can be mapped easily to the plain index sets $\{1, \cdots, m\}$ and $\{1, \cdots, n\}$.

We give two examples. In the first, the interaction kernel is $\alpha(t,s) = J_0(t-s)$, the zero-order Bessel function of the first kind, which appears in many two dimensional image processing problems [1]. The sources (targets) may be unequally spaced samples from a geometrically irregular source (target) domain. In the second example, the interaction kernel is $\alpha(t,s) = 1/\|t-s\|$, which we refer to as the Coulomb interaction, or gravitational interaction. This kernel appears frequently in the study of galaxy dynamics or molecular dynamics. The source and target points are the galactic or atomic bodies, respectively. In each of the examples, the matrix is dense and large.

We use a structure-adaptive indexing and tiling scheme for such kind of matrix-vector multiplication to expose the power-saving potential, among other purposes. We re-examine the conventional tiling scheme, i.e., the plain tiling as follows. (i) The indexing may be arbitrary, (ii) The tile partition is equally spaced, (iii) numerical values in each tile may vary in the whole range of the interaction function $\alpha(t,s)$ over the entire target-source domain, $\mathcal{T} \times \mathcal{S}$. (iv) The matrix is accessed row by row or column by column in tiles. To expose the potential for power saving, we break away from the above convention and convenience.

In our new scheme, which we refer to simply as the *geometric tiling*, we index and partition/tile the matrix elements according to the interaction kernel and the source-target distance. By doing so, we can claim a better estimate and control over the numerical range in each tile and across tiles. This is based on two underlying principles. The kernel functions, as in the two examples above, are continuous almost everywhere and invariant in spatial translation. These two properties are typical, not exceptional, in signal and image processing problems. Consider two source-target pairs close to each other, i.e., $\|t-t'\|$ and $\|s-s'\|$ are small. Then, $\alpha(t,s) \approx \alpha(t',s')$, by the continuity. Consider now two clusters, $\mathcal{T}' \subset \mathcal{T}$ and $\mathcal{S}' \subset \mathcal{S}$, with small enough geometric diameter. Then, the associated sub-matrix $A(\mathcal{T}', \mathcal{S}')$ has elements within a smaller numerical range, namely, the range of the interaction function on the sub-domain $\mathcal{T}' \times \mathcal{S}'$, instead of the entire target-source domain. We refer to such a sub-matrix as *a geometric tile*. We have thus translated the continuity into geometric tiles and claim the control over numerical range per tile by *geometric tiling*. We omit the algorithmic detail.

Next, we exploit the translation invariance property into geometric traversing. Let (t_1, s_1) and (t_2, s_2) be two source-target pairs again. If $\|t_1 - s_1\| \approx \|t_2 - s_2\|$, then $\alpha(t_1, s_1) \approx \alpha(t_2, s_2)$. In the same way, if the distance between clusters $\mathcal{S}_1$ and $\mathcal{T}_1$ is about the same as that between $\mathcal{S}_2$ and $\mathcal{T}_2$, then the numerical range of tile $A(\mathcal{T}_1, \mathcal{S}_1)$ is about the same as that of $A(\mathcal{T}_2, \mathcal{S}_2)$. Thus, all the tiles associated with interaction between clusters of the same distance are in the same numerical range. We may therefore traverse the matrix according to a sorting in the distance between clusters, to facilitate the architectural pipeline or data streaming in the same numerical range.

Figure 1 illustrates geometric tiling in a single row. In (a) there are a single target in bold point, and a set of source points in a two-dimensional sub-domain. It corresponds to a portion of a single row in the interaction matrix. And we assume the target is at the center of a target cluster. In (b) is a snapshot of an evaluation order with arbitrary indexing. The numerical variation may be large. In (c) is a much more desirable evaluation order as the numerical range is changing progressively from small to large or vice versa. In (d) we show a tiling among the sources (columns). The targets are tiled similarly.

II. PARALLELIZATION

Figure 2 gives the geometric tiling code under consideration. Note that, compared to the access pattern that one can expect from a plain matrix-vector multiplication, the access pattern exhibited by geometric tiling is more unconventional, not necessarily row or columnwise in tiles, following the geometric and point distribution we discussed in Section 1. Consequently, the data locality of a geometric tiling

*This work is supported in part by a grant from DARPA.

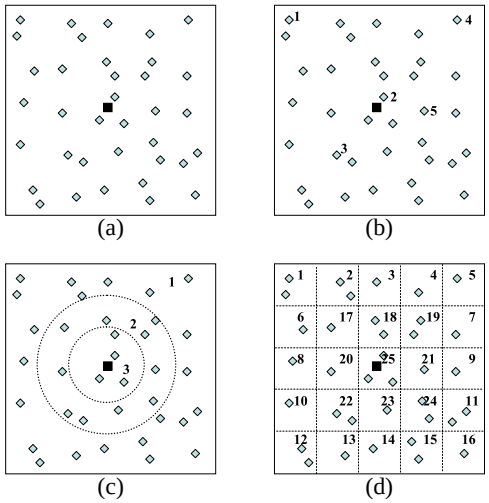

Figure 1: An illustration of geometric tiling in one row: one target in dark square and a subset of sources in diamonds. The integer index in each graph indicates an evaluation order. (a) a point distribution in a two-dimensional sub-domain. (b) an evaluation order in an arbitrarily given indexing, (c) bin-sorting the sources, and (d) geometric tiling among the sources.

```
for (targetBox=0;targetBox<L*L;targetBox++) {
  int *ii=TargetsInBox[targetBox];
  for (j=0;j<L;j++) {
    int dist=orderR[j];
    for (i=1;i<=InteractionList[targetBox][dist][0];i++){
      sourceBox=InteractionList[targetBox][dist][i];
      int *jj=SourcesInBox[sourceBox];
      for (m=1;m<=ii[0];m++) {
        int curii=ii[m];
        for (n=1;n<=jj[0];n++) {
          int curjj=jj[n];
          gr[curii][0]+=Ar[curjj][curii]*q[curjj][0];
          gr[curii][1]+=Ar[curjj][curii]*q[curjj][1];
        }
      }
    }
  }
}
```

Figure 2: Geometric tiling code.

code may not be as good as a plain matrix-vector multiplication code. However, we can still observe some data locality in the access pattern exhibited by geometric tiling. The Ar elements accessed by the computation involving the same source node and different targets fall within the same row of Ar, and therefore have better chance to fit in a smaller number of cache lines than those required when elements are scattered in different rows of Ar. In addition, the set of sources used is the same for all the targets in a box. Therefore, the accesses involving a target box and a source box pair should be together for the purpose of data locality. We now consider a parallelization strategy for the geometric tiling code presented in Figure 2. Our strategy tries to optimize the data locality within the execution of each processor. We parallelize the outermost loop, and each processor executes the operations on different set of target boxes. In this parallelization strategy. the accesses involving each target box and source box pair are grouped in a single processor, and consequently, for each processor, the data locality is optimized.

III. EXPERIMENTAL EVALUATION

The important point to note is that one may not need the full 32 bits for all the floating-point operations performed during geometric

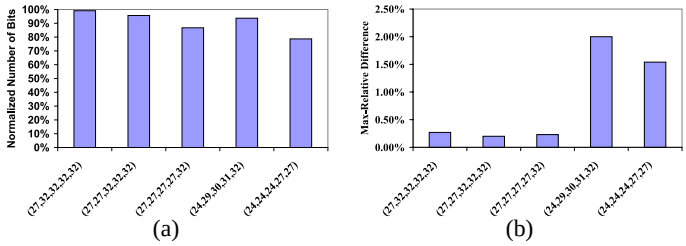

Figure 3: Bit savings and accuracy with different bit width configuration for floating-point operations. We assume a 5x5 geometric tiling. The numbers in each parenthesis pair give the number of bits for operations between box distances from 0 to 4 respectively. The base configuration is that 32 bits are used for all floating point operations. (a) Power consumption normalized with the base configuration. (b) Max-Relative difference compared to the base case.

tiling. Specifically, when the numerical value does not vary much in a tile, the tile can be represented in fewer bits without compromising numerical accuracy.

Therefore, some of the bits in a 32-bit adder can be shut down to save power. Figure 3(a) gives the normalized number of bits used (when considering all floating-point operations executed by the application) under five different configurations when simulating execution in SIMICS [2] for 8 processors, each with 8KB instruction/data caches and a 2MB shared unified L2 cache. Each point on the x-axis of this figure is represented by a quintuple, each entry of which gives the total number of bits used for a certain distance (we have five distance levels). It can be seen from this graph that we achieve significant bit savings in certain cases. For example, under the configuration (27,27,27,27,32), we are able to reduce the total number of bits used by around 13% (over the default cases which assumes 32 bits for each distance, i.e., the configuration (32,32,32,32,32)) and, using(24,24,24,27,27), we obtain a total bit saving of about 21%. One can expect these bit savings to lead to power savings as well provided that the underlying architecture supports shutting down the unused bit-lines in the floating-point unit. However, for a fair comparison, we need to check whether shutting down bits during geometric tiling leads to accuracy problems as well. Figure 3(b) gives the maximum relative difference between the reduced bit-width configuration and the full 32 bit configuration. It is clear from this figure that the first three configurations do not sacrifice much accuracy, while the last two configurations incur relatively higher inaccuracies (though the are still not very large).

IV. CONCLUSION

In this work, we propose, advocate and demonstrate the exploration of power-saving potential by exploiting application-specific characteristics. More specifically, focusing on a chip multiprocessor based execution environment, this paper proposes and experimentally evaluates a power-saving scheme, geometric tiling, which employs a non-traditional indexing, partition and traversing of a matrix to take advantage of the fact that the different tiles of the matrix can be processed using different bit-widths without affecting the accuracy.

V. REFERENCES

[1] M. Abramowitz and I. A. Stegun. *Handbook of mathematical functions, with formulas, graphs, and mathematical tables.* Published by National Bureau of Standards Applied Mathematics, 1972.

[2] SIMICS. http://www.virtutech.com/simics/simics.html.