Processor Lower Bound Formulas for Array Computations and Parametric Diophantine Systems

Peter Cappello and Ömer Eğecioğlu
Department of Computer Science
University of California at Santa Barbara
{omer, cappello}@cs.ucsb.edu

Abstract

Using a directed acyclic graph (dag) model of algorithms, we solve a problem related to precedence-constrained multiprocessor schedules for array computations: Given a sequence of dags and linear schedules parametrized by n, compute a lower bound on the number of processors required by the schedule as a function of n. In our formulation, the number of tasks that are scheduled for execution during any fixed time step is the number of non-negative integer solutions d_n to a set of parametric linear Diophantine equations. We illustrate an algorithm based on generating functions for constructing a formula for these numbers d_n. The algorithm has been implemented as a Mathematica program. An example run and the symbolic formula for processor lower bounds automatically produced by the algorithm for Gaussian Elimination *is presented.*

1. Introduction

We consider array computations, often referred to as systems of uniform recurrence equations. Parallel execution of uniform recurrence equations has been studied extensively, from at least as far back as 1966 (see, e.g., [15]). In such computations, the tasks to be computed are viewed as the nodes of a directed acyclic graph, where the data dependencies are represented as arcs. Given a dag $G = (N, A)$, a multiprocessor schedule assigns node v for processing during step $\tau(v)$ on processor $\pi(v)$. A valid multiprocessor schedule is subject to two constraints: 1) A node can be computed only after its predecessor nodes have been computed; 2) A processor cannot compute 2 different nodes during the same time step. We will refer to valid schedules simply as schedules. A schedule is good, if it uses time efficiently; an implementation of a schedule is good, if it uses few processors. This view prompted several researchers to investigate processor-time-minimal schedules for families of dags that represent fundamental problems (e.g. [1, 5, 6, 4]). These are time-minimal schedules that in addition use as few processors as possible. Clauss, Mongenet, and Perrin [2] developed a set of mathematical tools to help find a processor-time-minimal multiprocessor array for a given dag. Another approach to a general solution has been reported by Wong and Delosme [8], and Shang and Fortes [7]. They present methods for obtaining optimal linear schedules. That is, their processor arrays may be suboptimal, but they get the best linear schedule possible. Darte, Khachiyan, and Robert [11] show that such schedules are close to optimal, even when the constraint of linearity is relaxed. A lower bound on the number of processors needed to satisfy a schedule for a particular time step can be formulated as the number of solutions to a linear Diophantine equation, subject to the linear inequalities of the convex polyhedron that defines the dag's computational domain. The maximum processor bound for a given linear schedule, taken over all time steps, is a lower bound for the number of processors needed to satisfy the schedule for the dag family.

Such a geometric/combinatorial formulation for the study of a dag's task domain has been used in various other contexts in parallel algorithm design (see Fortes, Fu, and Wah [3] for a survey of systolic/array algorithm formulations). Here, we present a more general and uniform technique for deriving such lower bounds:

> *Given a parametrized dag family and a correspondingly parametrized linear schedule, we compute a* **formula** *for a lower bound on the number of processors required by the schedule.*

This is much more general than the analysis of an optimal schedule for a given *specific* dag. The lower bounds obtained are good; we know of no dag treatable by this method for which the lower bounds are not also upper bounds. We believe this to be the first reported algorithm and its implementation for automatically generating such formulae.

Augmenting the constraints defining the computational domain of the dag, by the linear constraint imposed by the schedule results in a linear Diophantine system of the form

$$\mathbf{a}\mathbf{z} = n\mathbf{b} + \mathbf{c}\,, \tag{1}$$

where the matrix $\mathbf{a}$ and the vectors $\mathbf{b}$ and $\mathbf{c}$ are integral, but not necessarily non-negative. The number d_n of solutions in non-negative integers $\mathbf{z} = [z_1, z_2, \ldots, z_s]^t$ to this linear system is a lower bound for the number of processors required when the dag corresponds to parameter n. Our algorithm produces (symbolically) the generating function for the sequence d_n, and from the generating function, a formula for the numbers d_n. We do not make use of any special properties of the system that reflects the fact that it comes from a dag. Thus in (1), $\mathbf{a}$ can be taken to be an arbitrary $r \times s$ integral matrix, and $\mathbf{b}$ and $\mathbf{c}$ arbitrary r-dimensional integral vectors. As such we actually solve a more general combinatorial problem of constructing the generating function $\sum_{n\geq 0} d_n t^n$, and a formula for d_n given a matrix $\mathbf{a}$ and vectors $\mathbf{b}$ and $\mathbf{c}$, for which the lower bound computation is a special case. There is a large body of literature concerning lattice points in convex polytopes and numerous interesting results: see for example Stanley [25] for Ehrhart polynomials, and Sturmfels [26, 27] for vector partitions and other mathematical treatments. Our results are based mainly on MacMahon [20, 21], and Stanley [24]. In the full paper [9], we describe the algorithm to construct the generating function and its proof, and remark on its implementation.

2. An Example: Gaussian Elimination

The algorithm for performing Gaussian elimination on an $n \times n$ matrix M below is taken from Golub and Van Loan [14].

```
for i = 0 to n − 1 do:
    for j = i + 1 to n − 1 do:
        w_j ← M[i, j];
    endfor;
    for j = i + 1 to n − 1 do:
        η ← M[j, i]/M[i, i];
        for k = i + 1 to n − 1 do:
            M[j, k] ← M[j, k] − η · w_j;
        endfor;
    endfor;
endfor;
```

We are interested in the triply-nested *for* loop, the heart of the computation. The computational nodes are defined by non-negative integral triplets (i, j, k) satisfying

$$\begin{aligned} i &\leq n-1 \\ i+1 \leq\ j &\leq n-1 \\ i+1 \leq\ k &\leq n-1 \end{aligned}$$

We eliminate the redundant constraint, yielding

$$\begin{aligned} i+1 \leq\ j &\leq n-1 \\ i+1 \leq\ k &\leq n-1 \end{aligned}$$

Our algorithm allows us to assume that the variables are non-negative. Introducing integral slack variables $s_1, s_2, s_3, s_4 \geq 0$, we obtain the equivalent linear Diophantine system

$$\begin{array}{rrrrrrrrrr} i & -j & & & +s_1 & & & & = & -1 \\ & j & & & & +s_2 & & & = n & -1 \\ i & & -k & & & & +s_3 & & = & -1 \\ & & k & & & & & +s_4 & = n & -1 \end{array}$$

A linear schedule for the corresponding dag is given by $\tau(i, j, k) = i + j + k + 1$. Since τ ranges from 1 to $3n - 2$, we can augment the system by adding the constraint at the halfway point: $\tau \approx \frac{3}{2}n - 1$. When n is an even number, say $n = 2N$, then we can take τ to be $3N - 1$. Adding the schedule constraint to system we already have, we obtain the augmented Diophantine system

$$\begin{array}{rrrrrrrrrr} i & +j & +k & & & & & & = 3N & -2 \\ i & -j & & & +s_1 & & & & = & -1 \\ & j & & & & +s_2 & & & = 2N & -1 \\ i & & -k & & & & +s_3 & & = & -1 \\ & & k & & & & & +s_4 & = 2N & -1 \end{array} \tag{2}$$

Here $\mathbf{b} = [3, 0, 2, 0, 2]^t$ and $\mathbf{c} = [-2, -1, -1, -1, -1]^t$. The system for Gaussian elimination for $n = 2N + 1$ is

$$\begin{array}{rrrrrrrrrr} i & +j & +k & & & & & & = 3N & -1 \\ i & -j & & & +s_1 & & & & = & -1 \\ & j & & & & +s_2 & & & = 2N & \\ i & & -k & & & & +s_3 & & = & -1 \\ & & k & & & & & +s_4 & = 2N & \end{array} \tag{3}$$

which differs from the even case only in the vector $\mathbf{c}$.

A lower bound for the number of processors needed to implement the schedule of the algorithm for Gaussian elimination without pivoting of an $n \times n$ matrix is the number of solutions of (2) if $n = 2N$, and the number of solutions of (3) if $n = 2N + 1$.

The final problem to be solved is the determination of the number of non-negative integral solutions d_n to a linear parametric Diophantine system of the form $\mathbf{a}\mathbf{z} = n\mathbf{b} + \mathbf{c}$ where $\mathbf{a}$ is a $r \times s$ integral matrix, $\mathbf{b}$ and $\mathbf{c}$ are r-dimensional integral vectors.

3. The General Formulation

We now generalize this example and consider the problem of computing a lower bound for the number of processors needed to satisfy a given linear schedule. That is, we

show how to automatically construct a formula for the number of lattice points inside a linearly parameterized family of convex polyhedra, by automatically constructing a formula for the number of solutions to the corresponding linearly parameterized system of linear Diophantine equations. The algorithm for doing this and its implementation are our principal contributions [9].

Our use of linear Diophantine equations, we believe, is well-motivated: the computations of an inner loop are typically defined over a set of indices that can be described as the lattice points in a convex polyhedron. Indeed, in two languages, SDEF [12] and ALPHA [28], one defines domains of computation explicitly as the integer points contained in some programmer-specified convex polyhedron.

The general setting exemplified by the *Gaussian Elimination* problem is as follows: Suppose $\mathbf{a}$ is an $r \times s$ integral matrix, and $\mathbf{b}$ and $\mathbf{c}$ are r-dimensional integral vectors. Suppose further that, for every $n \geq 0$, the linear Diophantine system $\mathbf{az} = n\mathbf{b} + \mathbf{c}$, i.e.

$$\begin{array}{ccccccccc} a_{11}z_1 & + & a_{12}z_2 & + & \ldots & + & a_{1s}z_s & = & b_1 n + c_1 \\ a_{21}z_1 & + & a_{22}z_2 & + & \ldots & + & a_{2s}z_s & = & b_2 n + c_2 \\ \vdots & & & & \vdots & & \vdots & = & \vdots \\ a_{r1}z_1 & + & a_{r2}z_2 & + & \ldots & + & a_{rs}z_s & = & b_r n + c_r \end{array} \tag{4}$$

in the non-negative integral variables $z_1, z_2, \ldots, z_s$ has a finite number of solutions. Let d_n denote the number of solutions for n. The generating function of the sequence d_n is $f(t) = \sum_{n \geq 0} d_n t^n$. For a linear Diophantine system of the form (4), $f(t)$ is always a rational function, and we provide an algorithm to compute $f(t)$ symbolically. The Mathematica program implementing the algorithm also constructs a formula for the numbers d_n from this generating function. Given a nested *for* loop, the procedure to follow is informally as follows:

1. Write down the node space as a system of linear inequalities. The loop bounds must be affine functions of the loop indices. The domain of computation is represented by the set of lattice points inside the convex polyhedron, described by this system of linear inequalities.

2. Eliminate unnecessary constraints by translating the loop indices (so that $0 \leq i \leq n-1$ as opposed to $1 \leq i \leq n$, for example). The reason for this is that the inequality $0 \leq i$ is implicit in our formulation, whereas $1 \leq i$ introduces an additional constraint.

3. Transform the system of inequalities to a system of equalities by introducing non-negative slack variables, one for each inequality.

4. Augment the system with a linear schedule for the associated dag, "frozen" in some intermediate time value: $\tau = \tau(n)$.

5. Run the program `DiophantineGF.m` on the resulting data. The program calculates the rational generating function $f(t) = \sum d_n t^n$, where d_n is the number of solutions to the resulting linear system of Diophantine equations, and produces a formula for d_n.

4. Sample Mathematica Run

Once the program `DiophantineGF.m` we have written for this computation[1] has been loaded by the command `<< DiophantineGF.m`, the user may request examples and help in its usage. The program essentially requires three arguments $\mathbf{a}, \mathbf{b}, \mathbf{c}$ of the Diophantine system (1). The main computation is performed by executing the command `DiophantineGF[a, b, c]`. The output is the (rational) generating function $f(t) = \sum_{n \geq 0} d_n t^n$, where d_n is the number of solutions $\mathbf{z} \geq \mathbf{0}$ to (1). After the computation of $f(t)$ by the program, the user can execute the command `formula`, which produces formulas for d_n in terms of binomial coefficients $C[x,k]$ (where $C[x,k] = 0$ if x is not integral) and in terms of the ordinary power basis in n when such a formula exists. The command `formulaN[c]` evaluates d_n for $n = c$. If needed, the generating function $f(t)$ computed by the program subsequently can be manipulated by various Mathematica commands, such as `Series[]`.

For Gaussian elimination without Pivoting of an $n \times n$ matrix the Diophantine system is $\mathbf{az} = N\mathbf{b} + \mathbf{c}$ where

$$\mathbf{a} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Here $\mathbf{b} = [3,0,2,0,2]^t$ and $\mathbf{c} = [-2,-1,-1,-1,-1]^t$, for $n = 2N$. The generating function for the number of non-negative integral solutions to (2) automatically computed by the program is

$$\frac{t^2(3+t)}{(1-t)^3(1+t)}.$$

The actual formula `DiophantineGF.m` produces for the coefficient of t^N in the expansion of this function is

$$\begin{array}{l} (3 * C[(N-2)/2, 0] - C[(N-4)/2, 0] - \\ 2 * C[(N-3)/2, 0])/8 \ + \\ (C[N-3, 2] + 3 * C[(N-2)/2, 0] - C[N-2, 2] - \\ 5 * C[N-1, 2] + 21 * C[N, 2])/8 \end{array} \tag{5}$$

[1] http://www.cs.ucsb.edu/∼omer/personal/abstracts/DiophantineGF.m

Since $C[x,0] = 0$ unless x is an integer, this means that

$$3 * C[(N-2)/2,0] - \quad C[(N-4)/2,0] - \\ 2 * C[(N-3)/2,0] \; = \; \begin{cases} 2 & N\ even, \\ -2 & N\ odd. \end{cases}$$

Simplifying the other binomial coefficients in (5), we get the lower bound for $n = 2N$ as

$$\frac{2N^2 - N}{2} \quad if\ N\ is\ even, \quad \frac{2N^2 - N - 1}{2} \quad if\ N\ is\ odd,$$

which can be combined into $\lfloor \frac{2N^2-N}{2} \rfloor$ for $n = 2N$. The system for Gaussian elimination for $n = 2N + 1$ is given in (3). In this case $\mathbf{c} = [-1, -1, 0, -1, 0]^t$ and $\mathbf{A}$ and $\mathbf{b}$ are the same as above. The generating function computed by the program is

$$\frac{t(1+3t)}{(1-t)^3(1+t)}.$$

Simplifying the automatically produced formula as before, we obtain

$$\frac{2N^2 - N}{2} \quad if\ N\ is\ even, \quad \frac{2N^2 - N - 1}{2} \quad if\ N\ is\ odd.$$

Therefore the lower bound for $n = 2N+1$ is also $\lfloor \frac{2N^2-N}{2} \rfloor$. Combining with the previous case, we obtain the processor lower bound

$$\lfloor \frac{\lfloor \frac{n}{2} \rfloor (2 \lfloor \frac{n}{2} \rfloor - 1)}{2} \rfloor$$

for Gaussian elimination without pivoting of an $n \times n$ matrix for arbitrary n.

5 Computational Complexity and Conclusion

The algorithm we provide for the computation of the generating function has a worst case exponential running time. This is not surprising; the simpler computation: "Are *any* processors scheduled for a particular time step?", which is equivalent to "Is a particular coefficient of the generating function non-zero?" is already known to be an NP-complete problem [22, 13]. However, such a formula needs to be constructed only once for a given algorithm. In practice, array algorithms typically have a description that is sufficiently succinct to make this automated formula production feasible.

To summarize the main ideas of this paper: given a nested loop program whose underlying computation dag has nodes representable as lattice points in a convex polyhedron, and a multiprocessor schedule for these nodes that is linear in the loop indices, we produce a formula for the number of lattice points in the convex polyhedron that are scheduled for a particular time step (which is a lower bound on the number of processors needed to satisfy the schedule). This is done by constructing a system of parametric linear Diophantine equations whose solutions represent the lattice points of interest. Our principal contribution is devising an algorithm and its implementation for constructing the generating function from which a formula for the number of these solutions is produced.

The example illustrated the relationship between nested loop programs and Diophantine equations, and was annotated with the output of a Mathematica program that implements the algorithm. The algorithm's exponential computational complexity should be seen in light of two facts:

- Deciding if a time step has *any* nodes associated with it is NP-complete; we construct a *formula* for the number of such nodes;
- This formula is a processor lower bound, not just for one instance of a scheduled computation but for a parameterized family of such computations.

In bounding the number of processors needed to satisfy a linear multiprocessor schedule for a nested loop program, we actually derived a solution to a more general linear Diophantine problem of the type given by (4). This leaves open some interesting combinatorial questions of rationality and algorithm design for systems similar to (4) where the right hand side consists of higher degree polynomials in n.

References

[1] A. Benaini and Yves Robert. Space-time-minimal systolic arrays for Gaussian elimination and the algebraic path problem. *Parallel Computing*, 15:211–225, 1990.

[2] Ph. Clauss, C. Mongenet, and G. R. Perrin. Calculus of space-optimal mappings of systolic algorithms on processor arrays. In *Proc. Int. Conf. on Application Specific Array Processors*, pages 4–18, Princeton, September 1990. IEEE Computer Society.

[3] José A. B. Fortes, King-Sun Fu, and Benjamin W. Wah. Systematic design approaches for algorithmically specified systolic arrays. In Veljko M. Milutinović, editor, *Computer Architecture: Concepts and Systems*, chapter 11, pages 454–494. North-Holland, Elsevier Science Publishing Co., New York, NY 10017, 1988.

[4] Chris Scheiman and Peter Cappello. A processor-time minimal systolic array for the 3d rectilinear mesh. In *Proc. Int. Conf. on Application Specific Array Processors*, pages 26–33, Strasbourg, FRANCE, July 1995.

[5] Chris Scheiman and Peter Cappello. A processor-time minimal systolic array for transitive closure. *IEEE*

Trans. on Parallel and Distributed Systems, 3(3):257–269, May 1992.

[6] Chris J. Scheiman and Peter Cappello. A period-processor-time-minimal schedule for cubical mesh algorithms. *IEEE Trans. on Parallel and Distributed Systems*, 5(3):274–280, March 1994.

[7] Weijia Shang and José A. B. Fortes. Time optimal linear schedules for algorithms with uniform dependencies. *IEEE Transactions on Computers*, 40(6):723–742, June 1991.

[8] Yiwan Wong and Jean-Marc Delosme. Space-optimal linear processor allocation for systolic array synthesis. In V.K. Prasanna and L. H. Canter, editors, *Proc. 6th Int. Parallel Processing Symposium*, pages 275–282. IEEE Computer Society Press, Beverly Hills, March 1992.

[9] Peter Cappello and Ömer Eğecioğlu. Processor lower bound formulas for array computations and parametric Diophantine systems. http://www.cs.ucsb.edu/∼omer/personal/Combinatorics.shtml

[10] Peter Cappello and Kenneth Steiglitz. Unifying VLSI array design with linear transformations of space-time. In Franco P. Preparata, editor, *Advances in Computing Research*, volume 2: VLSI theory, pages 23–65. JAI Press, Inc., Greenwich, CT, 1984.

[11] Alain Darte, Leonid Khachiyan, and Yves Robert. Linear scheduling is close to optimal. In José Fortes, Edward Lee, and Teresa Meng, editors, *Application Specific Array Processors*, pages 37–46. IEEE Computer Society Press, August 1992.

[12] Bradley R. Engstrom and Peter Cappello. The SDEF programming system. *J. of Parallel and Distributed Computing*, 7:201–231, 1989.

[13] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979.

[14] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*, 2nd Edition, The Johns Hopkins University Press, Baltimore, 1990.

[15] Richard M. Karp, Richard E. Miller, and Shmuel Winograd. Properties of a model for parallel computations: Determinacy, termination, queueing. *SIAM J. Appl. Math*, 14:1390–1411, 1966.

[16] H.-T. Kung and Charles E. Leiserson. Algorithms for VLSI processor arrays. In *Introduction to VLSI Systems*, pages 271–292. Addison-Wesley Publishing Co, Menlo Park, CA, 1980.

[17] Percy A. MacMahon. Memoir on the Theory of the Partitions of Numbers- Part II, in *Collected Papers, Vol. I, Combinatorics*, George E. Andrews, Editor, The MIT Press, Cambridge, MASS, 1978, pp. 1138–1188.

[18] Percy A. MacMahon. Memoir on the Theory of the Partitions of Numbers- Part IV, in *Collected Papers, Vol. I, Combinatorics*, George E. Andrews, Editor, The MIT Press, Cambridge, MASS, 1978, pp. 1292–1314.

[19] Percy A. MacMahon. Application of the Partition Analysis to the study of the properties of any system of consecutive integers. in *Collected Papers, Vol. I, Combinatorics*, George E. Andrews, Editor, The MIT Press, Cambridge, MASS, 1978, pp. 1189–1211.

[20] Percy A. MacMahon. The Diophantine Inequality $\lambda x \geq \mu y$. in *Collected Papers, Vol. I, Combinatorics*, George E. Andrews, Editor, The MIT Press, Cambridge, MASS, 1978, pp. 1212–1232.

[21] Percy A. MacMahon. Note on the The Diophantine Inequality $\lambda x \geq \mu y$. in *Collected Papers, Vol. I, Combinatorics*, George E. Andrews, Editor, The MIT Press, Cambridge, MASS, 1978, pp. 1233–1246.

[22] Sartaj Sahni. Computational related problems. *SIAM J. Comput.*, 3:262–279, 1974.

[23] Chris Scheiman. *Mapping Fundamental Algorithms onto Multiprocessor Architectures*. Ph.D. Thesis, UC Santa Barbara, Dept. of Computer Science, Dec., 1993.

[24] Richard P. Stanley Linear homogeneous diophantine equations and magic labelings of graphs. Duke Math. J., 40:607-632, 1973.

[25] Richard P. Stanley. *Enumerative Combinatorics, Volume I*, Wadsworth & Brooks/Cole, Monterey, CA 1986.

[26] Bernd Sturmfels. *Gröbner Bases and Convex Polytopes*, AMS University Lecture Series, Providence, RI 1995.

[27] Bernd Sturmfels. *On Vector Partition Functions*, J. Combinatorial Theory, Series A, 72:302–309, 1995.

[28] H. Le Verge, C. Mauras, and P. Quinton. The ALPHA language and its use for the design of systolic arrays. *J. VLSI Signal Processing*, 3:173–182, 1991.