Berkowitz Algorithm in parallel with the Maple Grid Computing Toolbox

Gema Mª Díaz-Toca, Alfonso López Murcia

Universidad de Murcia, Spain, gemadiaz@um.es, alfonso@um.es

Goal: To compute the characteristic polynomial of a polynomial matrix.

Possible Algorithms

- Le Verrier Algorithm,
- Souriau-Faddeev-Frame Algorithm,
- Preparata-Sarwate Algorithm,
- Chistov Algorithm,
- Berkowitz Algorithm.

The Chosen: Berkowitz Algorithm

Notation

Let $A = (a_{ij}) \in \mathbb{Q}[\overline{x}]_{n\times n}$.

- $\mathbb{I}_r$: the identity $r \times r$ matrix;
- A_r: the leading principal submatrix of order r of A;
- $P_r(\lambda) = \det(\lambda\mathbb{I}_r - A_r) = \sum_{i=0}^{r} c_i\lambda^i$ the characteristic polynomial of A_r;
- R_r: the r row vector of elements $a_{r+1,j}$ such that $1 \le j \le r$ (here $r \le n-1$) ;
- S_r: the r column vector of elements $a_{i,r+1}$ such that $1 \le i \le r$ (here $r \le n-1$).
- Given $P(\lambda) = \sum_{k=0}^{d} a_k\,\lambda^k$, and

$$\overrightarrow{P} = \begin{pmatrix} a_d \\ a_{d-1} \\ \vdots \\ a_0 \end{pmatrix},$$

let Toep(P) denote a $(d+1)\times d$ subdiagonal Toeplitz matrix associated to the coefficients of P:

$$\text{Toep}(P) = \begin{pmatrix} a_d & 0 & \cdots & 0 \\ a_{d-1} & a_d & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & \cdots & a_d \\ a_0 & a_1 & \cdots & a_{d-1} \end{pmatrix}.$$

The key: Samuelson's Formula

Consider the following partition of A_{r+1} *:*

$$A_{r+1} = \begin{pmatrix} A_r & S_r \\ R_r & a_{r+1,r+1} \end{pmatrix},$$

let $P_r(\lambda) = c_r\lambda^r + c_{r-1}\lambda^{r-1} + \ldots + c_0$ *. Then*

$$\overrightarrow{P_{r+1}} = \text{Toep}\,(Q_{r+1}) \times \overrightarrow{P_r}$$

where Q_{r+1} *is*

$$Q_{r+1} = \lambda^{r+1} - a_{r+1,r+1}\lambda^r - \sum_{i=0}^{r-1} R_r A_r^i S_r \lambda^{r-1-i}.$$

So, **the characteristic polynomial of** A **is**

$$\overrightarrow{P_n} = \text{Toep}\,(Q_n) \times \text{Toep}\,(Q_{n-1}) \times \cdots \times \text{Toep}\,(Q_1).$$

As a result,

Sequential Berkowitz Algorithm[1]

Input: An n–square matrix $A \in \mathbb{D}^{n\times n}$.
Output: Characteristic polynomial of A.
(SBA.1) **Initialize** the vector Vect to

$$\text{Vect} := \begin{pmatrix} 1 \\ -a_{11} \end{pmatrix}.$$

(SSBA.2) **for** r **from** 1 **to** $n-1$,
(SBA.2.1) **Compute** the entries $\{R_r A_r^{k-1} S_r\}_{k=1}^{r}$ of the Toeplitz matrix Toep$\,(Q_{r+1})$;
(SBA.2.2) **Update** Vect into Vect := Toep$\,(Q_{r+1}) \times$ Vect ;
(SBA.3) **Return** $\overrightarrow{P_n}$ = Vect.

Parallel study

- Toeplitz matrices Toep$\,(Q_{r+1})$ are independent. So, (SBA.1) can be done in parallel.
- The characteristic polynomial $\overrightarrow{P_n}$ is provided by a product. This product can be done in parallel too.

The Tools

- Workstation with a Intel Pentium Quad Core processor.
- Maple 11, a comercial mathematics software for symbolic computation.

However, Maple 11 does not run in parallel. To parallelize Berkowitz Algorithm, there are two options:

1. Using OpenMaple API with MPI.
2. Buying the Grid Computing Toolbox for Maple.

OpenMaple is a suite of functions that allows you to access Maple algorithms and data structures in C, Java or Visual Basic programs. We transcribed Maple Berkowitz sequential code to a C program. Disadvantages:

- C code is complex.
- A lot of data type conversions are needed (C to Maple and viceversa).
- If Maple garbage collection runs, some variable values are cleaned and the program crashes. It's necessary to protect variables.
- Poor performance, times are high vs Maple times.

If we append MPI API to C code, there is no improvement. Therefore, we discard OpenMaple API with MPI.

Grid Computing Toolbox for Maple is a Maple Library that contains procedures for distributing computations across an arbitrary number of machines and/or CPUs on the same machine.

Grid Computing Toolbox

Grid Computing Toolbox offers MPI-like commands for message passing.

Command	Action
Send	Send a message
Receive	Receive a message
Seq	Sequence over grid
Map	Map over grid

But structures with more than one dimension (like a Matrix) are not currently supported in Seq and Map commands. We only use Send and Receive commands like a MPI program.

First Parallel Algorithm

Load A from a file.
A is in all processors.
P_0: **for** i **from** 2 **to** $nproc$
Calculate index_toep_list;
msg:=index_toep_list;
Send(i-1,msg);
P_0: **for** i **from** 2 **to** n
msg:=**Receive**();
position:=op(1,msg);
result:=op(2,msg);
toeplist[position]:=result;
P_0: **for** i **from** 2 **to** $nproc$
Calculate start and end of group of matrices;
toepgroup:=[op(start..end,toeplist)];
Send(i-1,toepgroup);
P_0: Q:=toepmatrixmult(1,myend);
P_0: **for** i **from** 2 **to** $nproc$
result:=**Receive**(i-1);
C:=op(2,result).Q;
Q:=C;
P_0: polcar:=add$(Q[i+1] * X^(n-i), i = 0..n)$;
$P_{1..3}$: msg := **Receive**(0);
$P_{1..3}$: **for** i **from** 1 **to** $nops(msg)$
position:=op(1,msg);
result:=toep(position);
Send(0,[position,result]);
$P_{1..3}$: matrixgroup:=**Receive**(0);
$P_{1..3}$: result:=toepmatrixmult(matrixgroup);
$P_{1..3}$: **Send**(0,[thisnode,result]);

Note

To start with parallel computation, Grid command `Launch(nodes,code,printf,checkAbort,["A"])` imports A to each of the nodes.

Future work

To compare parallel time with sequential time.
To optimize parallel algorithm.

References

[1] J. Abdeljaoued and H. Lombardi, *Méthodes Matricielles. Introduction à la Complexité Algébrique.* Springer (2004).

Acknowledgments. This work has been partially supported by the Department of Applied Mathematics and by MEC grant MTM2005-08690-C02-02.