Improving linear algebra in Jorek

Anemos meeting in Nice

X. Lacoste, P. Ramet

Xavier LACOSTE
HiePACS team
Inria Bordeaux Sud-Ouest

Guideline

Introduction

Assembly in Jorek
- Original assembly description
- Using MURGE in Jorek

Results

Current works in PaStiX

Conclusion

1
Introduction

Introduction

Goals

- Reduce memory peak;
- Reduce computation time.

2
Assembly in Jorek

Assembly in Jorek (original version)

// Each processor is in charge of a distinct part of the element_list
for each element e in *local_elements_list*
    $EM = elementMatrix(e)$
    for each vertex v_1 of e
        for each order o_1
            for each i ¡ $n_{tor} * n_{var}$
                $X = computeGlobalCoordinate(e, v_1, o_1, i)$
                $x = n_{tor} * n_{var} * (n_{order} + 1) * (v_1 - 1) +$
                    $n_{tor} * n_{var} * (o_1 - 1) + i$
                for each vertex v_2 of e
                    for each order o_2
                    for each j ¡ $ntor * nvar$
                        $Y = computeGlobalCoordinate(e, v_2, o_2, j)$
                        $y = n_{tor} * n_{var} * (n_{order} + 1) * (v_2 - 1) +$
                            $n_{tor} * n_{var} * (o_2 - 1) + j$
                        $DistributedMatProd(X, Y) = EM(x, y)$
$FullMatHarm = DistributeHarmonics(DistributedMatProd)$

Detail of an elementary matrix

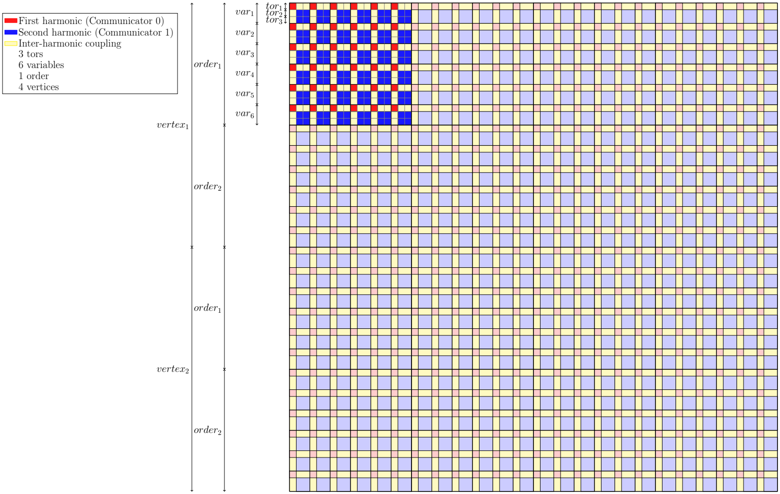

Figure: Part of an elementary matrix

Detail of the assembly

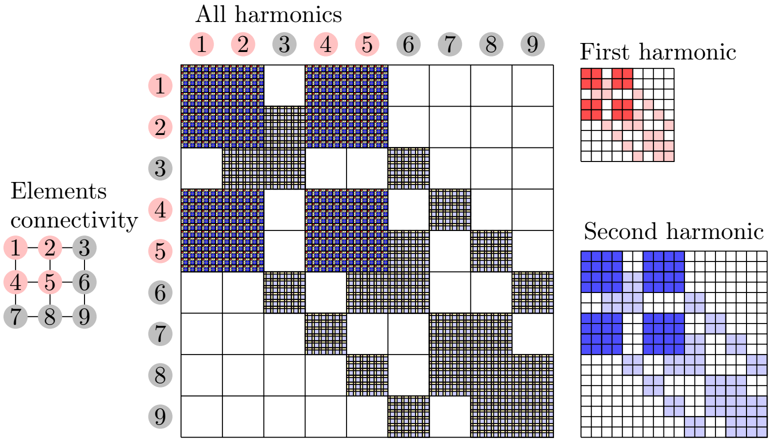

Figure: Assembly of an element E (vertices={1,2,,5})

Assembly in Jorek (MURGE version)

```
// An element is local if it contributes to local columns of PaStiX Matrix (harmonic)
// The obtained list is distributed among inter-harmonic-communicator
for each element e in local_elements_list
    EM = elementMatrix(e)
    for each vertex v1 of e
        for each order o1
            for each i ¡ n_tor * n_var
                X = computeGlobalCoordinate(e, v1, o1, i)
                X_harm = computeHarmCoordinate(e, v1, o1, i)
                x = n_tor * n_var * (n_order + 1) * (v1 − 1)+
                    n_tor * n_var * (o1 − 1) + i
                for each vertex v2 of e
                    for each order o2
                    for each j ¡ ntor * nvar
                        Y = computeGlobalCoordinate(e, v2, o2, j)
                        Y_harm = computeHarmCoordinate(e, v2, o2, j)
                        y = n_tor * n_var * (n_order + 1) * (v2 − 1)+
                            n_tor * n_var * (o2 − 1) + j
                        DistributedMatProd(X, Y) = EM(x, y)
                        if (harm)
                            if (local_harm)
                                DistributedMatHarm(X_harm, Y_harm) = EM(x, y)
                            else
                                SendHarm(X_harm, Y_harm, EM(x, y))
```

GMRES Workflow

Once	Preprocessing	PT-Scotch on a compressed graph (1 node for ntor*nvar entries)
Every iteration	Assembly	Distributed assembly MPI + OpenMP, centralisation of the matrix for PaStiX call/MURGE works on distributed local harmoniques
If needed	Factorization	Factorisation PaStiX MPI/P-Threads
Every iteration	GMRES	Solve and `Ax` product with PaStiX MPI/P-Threads

Advantages

- Save time :
  - Compressed graph in Scotch (can be activated both with MURGE and PaStiX);
  - Use parallel PT-Scotch with distributed graph in MURGE;
  - Matrix directly build in the correct format, IJV to CSC translation is not neglectable;
- Save memory :
  - Compressed and distributed graph;
  - Harmonic matrix is distributed and stored only once (whereas it is stored twice (IJV and CSC) when using PaStiX);
  - Distributed CSC is smaller than distributed IJV for the global matrix;
- More abstraction :
  - Communications and computations to create PaStiX CSCd matrix are hidden;
  - Storage of the matrix is hidden $\Rightarrow$ any other solver using MURGE interface could easily be used instead of PaStiX.

3
Results

Few results on model 302, 4 nodes, 8 threads

	constr	dist	preproc	fact	gmres/solve	total
PaStiX	8.39 s	8.79 s	19/114 s	6.68/34.3 s	1-9 s	187 s
PaStiX (compressed)	9.48 s	9.11 s	9/10 s	4.95/34.2 s	2-13 s	75.7 s
MURGE	14.4 s	–	0.20/0.21s s	3.84/24.8 s	1-6 s	49.2 s

Table: Iteration with factorization

	constr	dist	preproc	fact	gmres/solve	total
PaStiX	7.70 s	–	–	–	1-9 s	11.4 s
PaStiX (compressed)	8.40 s	–	–	–	2-13 s	13.0 s
MURGE	9.60 s	–	–	–	1-6 s	12.5 s

Table: Iteration without factorization

4
Current works in PaStiX

Developpement realised to improve Jorek

- Possibility to use thread funneled and thread multiple modes in the same run (eg. funneled factorization and multiple resolution);
- MURGE: possibility to register a sequence of entries to perform next similar assembly faster;
- MURGE: possibility to build matrix with one column distributed on several processors if only a product is performed.

PaStiX current works

Sparse linear solver over runtime

- Rewrite factorization and solve steps on top of StarPU and ParSEC;
- Goal : obtain similar performances to classical PaStiX with share memory runs;
- Possibility of using other computing units for "free";
- Possibility of chaining factorization and solve.

5
Conclusion

Conclusion

Improvement on linear Algebra in Jorek in term of both time and memory.

Futur works

- Compare the time cost of matrix construction with the two methods;
- Compare the memory cost of matrix construction with the two methods;
- Improve assembly time with MURGE.
  - Compare elementary matrix building duplication versus communication;

Thanks !

Xavier LACOSTE
INRIA HiePACS team

Anemos Meeting – February 01, 2013