Solveurs Directs : Introduction à PaStiX

Formation en Algèbre Linéaire Creuse Parallèle - Bordeaux, novembre 2011

Pierre Ramet

30 novembre 2011

Présentation

PaStiX est un solveur de systèmes linéaires creux développé à l'INRIA Bordeaux - Sud-Ouest. Il est disponible sur la forge de l'INRIA.

- **Site :** `http://pastix.gforge.inria.fr`
- **Forum :** `http://gforge.inria.fr/forum/?group_id=186`
- **Liste de diffusion :**
  `pastix-users@lists.gforge.inria.fr`

```
module load mds-intel-openmpi mds-scotch/5.1.11
mds-pastix/5.1.4
```

Bibliothèques nécessaires

- Une bibliothèque de communication MPI :
  Mpich2, OpenMPI, ...
- Une bibliothèque de BLAS :
  GotoBLAS, ACML, MKL, ...
- La bibliothèque Scotch pour la renumérotation des inconnues
  (la bibliothèque MetiS est facultative).

Compilateur et options de compilation

```
HOSTARCH    = i686_mac
VERSIONBIT  = _32bit
EXEEXT      =
OBJEXT      = .o
LIBEXT      = .a
CCPROG      = gcc -Wall
CFPROG      = gfortran
CF90PROG    = gfortran
MCFPROG     = mpif90
CF90CCPOPT  = -xf95-cpp-input
# Compilation options for optimization (make expor)
CCFOPT      = -O3
# Compilation options for debug (make | make debug)
CCFDEB      = -g3

LKFOPT      =
MKPROG      = make
MPCCPROG    = mpicc -Wall
CPP         = cpp
ARFLAGS     = ruv
ARPROG      = ar
EXTRALIB    = -lgfortran -lm
```

Type d'entiers et type d'arithmétique

```
#VERSIONINT  = _long
#CCTYPES     = -DFORCE_LONG -DLONG
#--------------------------
#VERSIONINT  = _int32
#CCTYPES     = -DFORCE_INT32 -DINTSIZE32
#--------------------------
#VERSIONINT  = _int64
#CCTYPES     = -DFORCE_INT64 -DINTSSIZE64

# uncomment the following lines for double precision support
VERSIONPRC  = _double
CCTYPESFLT := $(CCTYPESFLT) -DFORCE_DOUBLE -DPREC_DOUBLE

# uncomment the following lines for float=complex support
#VERSIONFLT  = _complex
#CCTYPESFLT := $(CCTYPESFLT) -DFORCE_COMPLEX -DTYPE_COMPLEX
```

Support MPI/threads

```
# uncomment the following lines for sequential (NOMPI) version
#VERSIONMPI  = _nompi
#CCTYPES    := $(CCTYPES) -DFORCE_NOMPI
#MPCCPROG    = $(CCPROG)
#MCFPROG     = $(CFPROG)

# uncomment the following lines for non-threaded (NOSMP) version
#VERSIONSMP  = _nosmp
#CCTYPES    := $(CCTYPES) -DFORCE_NOSMP

# Uncomment the following line to enable a progression thread
#CCPASTIX   := $(CCPASTIX) -DTHREAD_COMM

# Uncomment the following line if your MPI doesn't support MPI_THREAD_MULTIPLE level
#CCPASTIX   := $(CCPASTIX) -DPASTIX_FUNNELED

# Uncomment the following line if your MPI doesn't support MPI_Datatype correctly
#CCPASTIX   := $(CCPASTIX) -DNO_MPI_TYPE
```

Options du solveur

```
# Uncomment the following lines for NUMA-aware allocation (recommended)
CCPASTIX   := $(CCPASTIX) -DNUMA_ALLOC

# Show memory usage statistics
#CCPASTIX   := $(CCPASTIX) -DMEMORY_USAGE

# Show memory usage statistics in solver
#CCPASTIX   := $(CCPASTIX) -DSTATS_SOPALIN

# Uncomment following line for dynamic thread scheduling support
#CCPASTIX   := $(CCPASTIX) -DPASTIX_DYNSCHED

# Uncomment the following lines for Out-of-core
#CCPASTIX   := $(CCPASTIX) -DOOC -DOOC_NOCOEFINIT -DOOC_DETECT_DEADLOCKS
```

Bibliothèques annexes (1/2)

```
# uncomment the following lines for using metis ordering
#VERSIONORD  = _metis
#METIS_HOME  = ${HOME}/metis-4.0
#CCPASTIX   := $(CCPASTIX) -DMETIS  -I$(METIS_HOME)/Lib
#EXTRALIB   := $(EXTRALIB) -L$(METIS_HOME) -lmetis

# Scotch always needed to compile
SCOTCH_HOME ?= ${HOME}/Work/scotch/
SCOTCH_INC ?= $(SCOTCH_HOME)/include
SCOTCH_LIB ?= $(SCOTCH_HOME)/lib
# uncomment on of this blocks
#scotch
#CCPASTIX   := $(CCPASTIX) -I$(SCOTCH_INC) -DWITH_SCOTCH
#EXTRALIB   := $(EXTRALIB) -L$(SCOTCH_LIB) -lscotch -lscotcherrexit
#ptscotch
CCPASTIX   := $(CCPASTIX) -I$(SCOTCH_INC) -DDISTRIBUTED -DWITH_SCOTCH
EXTRALIB   := $(EXTRALIB) -L$(SCOTCH_LIB) -lptscotch -lscotcherrexit
```

Bibliothèques annexes (2/2)

```
# Choose Blas library (Only 1)
# Do not forget to set BLAS_HOME if it is not in your environnement
# BLAS_HOME=/path/to/blas
#----  Blas    ----
BLASLIB =  -lblas
#---- Gotoblas ----
#BLASLIB =  -L$(BLAS_HOME) -lgoto
#----  MKL     ----
# Uncomment the correct line
#BLASLIB =  -L$(BLAS_HOME) -lmkl_intel_lp64 -lmkl_sequential -lmkl_core
#BLASLIB =  -L$(BLAS_HOME) -lmkl_intel -lmkl_sequential -lmkl_core
#----  Acml    ----
#BLASLIB =  -L$(BLAS_HOME) -lacml
```

Attention

Ne pas activer le support des threads GotoBLAS ou MKL

Tests

Compiler PaStiX

`make clean`	Nettoie le répertoire
`make expor`	Compile la bibliothèque en mode optimisé
`make debug`	Compile la bibliothèque en mode debug
`make install`	Installe la bibliothèque
`make examples`	Compile les exemples en C et en Fortran

Tests

Essayer les différents exemples de PaStiX avec les matrices disponibles : `module load mds-matrix`.

Appeler PaStiX depuis un code *utilisateur*? (1/2)

```
void pastix(Pastix_Data_t **pastix_data,
            MPI_Comm        pastix_comm,
            pastix_int_t    n,
            pastix_int_t   *colptr,
            pastix_int_t   *row,
            pastix_float_t *avals,
            pastix_int_t   *perm,
            pastix_int_t   *invp,
            pastix_float_t *b,
            pastix_int_t    rhs,
            pastix_int_t   *iparm,
            double         *dparm);
```

Appeler PaStiX depuis un code *utilisateur* ? (2/2)

3 fichiers sont créés lors de l'étape `make install` :

- `pastix.h`
- `pastix_fortran.h`
- `libpastix.a`

Pour ne pas oublier les bibliothèques nécéssaires à PaStiX, utilisez la commande (`pastix-conf`)

Les paramètres iparm/dparm

Affichage

API_VERBOSE_NOT	0	Aucun affichage
API_VERBOSE_NO	1	Affichage limité
API_VERBOSE_YES	2	Affichage complet

Critères de terminaison

IPARM_ITERMAX	5
DPARM_EPSILON_REFINEMENT	5

Type de factorisation

IPARM_FACTORIZATION	30	0 : LL^T, 1 : LDL^T, 2 : LU
IPARM_SYM	40	API_SYM_YES ou API_SYM_NO

Exécution étape par étape

1. Initialisation de la structure du solveur
2. Renumérotation
3. Factorisation symbolique
4. Distribution et calcul de l'ordonnancement
5. Factorisation
6. Résolution
7. Raffinement
8. Libération des structures

Regardez le code de l'exemple `Step-by-Step`.

Parallélisme de thread

Intérêts

- Meilleure scalabilité
- Meilleure utilisation des nouvelles architectures multi-cœurs
- Consommation mémoire plus faible (moins de buffers de communication)

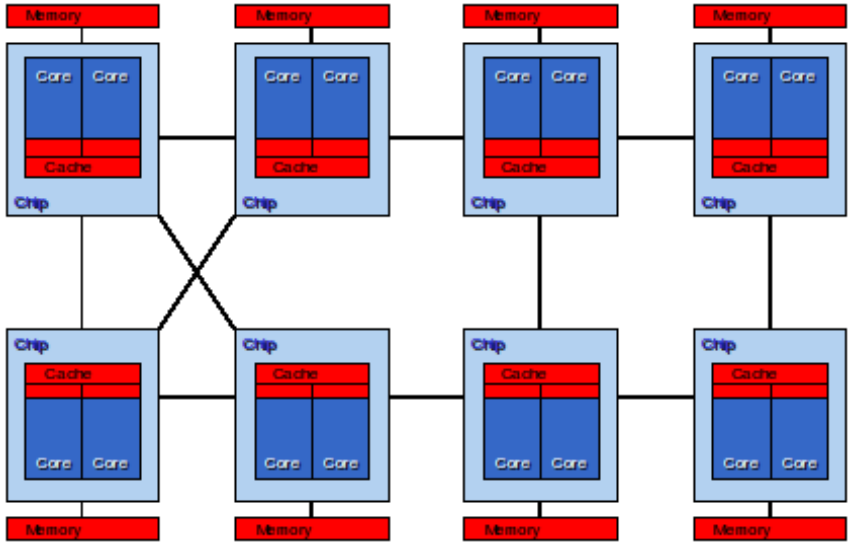
Comment ?

Utilisation du paramètre `IPARM_THREAD_NBR`

Regardez la différence de surcoût mémoire entre deux exécutions.
Par exemple :

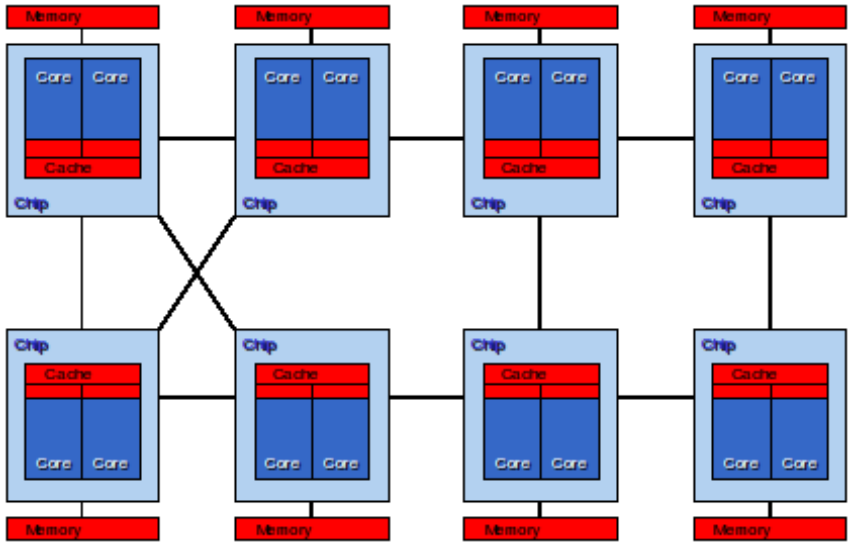
- 8 processus MPI de 2 threads
- 4 processus MPI de 4 threads
- 2 processus MPI de 8 threads

Bien placer ses threads (1/2)



- Importance du placement sur les nouvelles architectures
- Attention aux problèmes de concurrence

Bien placer ses threads (2/2)



pastix_setBintab

```
void pastix_setBindtab(
                Pastix_Data_t **pastix_data,
                int             thrdnbr,
                int            *bindtab)
```

Comparez une version où les threads sont correctement répartis et une où vous affectez volontairement plusieurs threads à un même cœur.