VERROU: localisation temporelle non intrusive

14/06/24

Bruno Lathuilière (EDF R&D)

Réunion Interflop

Verrou : état d'avancement

- InterlibM:
  - Tous les modes d'arrondis du backend verrou (y compris les versions déterministe/stochastique)
  - Automaticité pour l'utilisateur (`-libm=<auto_exclude|manual_exclude|instrumented>`)
  - Compatibilité avec le delta-debug
    - gestion espace de recherche et du status d'instrumentation (via client request)
    - choix de la référence natif/nearest
  - Détection Nan et Inf
  - Manque les fonctions complexes
- Suppression de la dé-vectorisation dans le frontend
- Portage ARM64:
  - Limitations: pas d'intégration continue, backend mcaquad déactivé.
  - Opérations fp16 non implémentées
- Localisation temporelle:
  - Mise au propre de l'instrumentation temporelle instrusive
  - Mise au propre du delta-debug python
  - Mise au point de la localisation temporelle non-intrusive

Verrou : état d'avancement

- InterlibM:
  - Tous les modes d'arrondis du backend verrou (y compris les versions déterministe/stochastique)
  - Automaticité pour l'utilisateur (`-libm=<auto_exclude|manual_exclude|instrumented>`)
  - Compatibilité avec le delta-debug
    - gestion espace de recherche et du status d'instrumentation (via client request)
    - choix de la référence natif/nearest
  - Détection Nan et Inf
  - Manque les fonctions complexes
- Suppression de la dé-vectorisation dans le frontend
- Portage ARM64:
  - Limitations: pas d'intégration continue, backend mcaquad déactivé.
  - Opérations fp16 non implémentées
- **Localisation temporelle:**
  - Mise au propre de l'**instrumentation temporelle instrusive**
  - Mise au propre du delta-debug python
  - Mise au point de la **localisation temporelle non-intrusive**

Muller exemple

```
float muller(int nt, bool verbose=false){
   float x0 = 11./2.;
   float x1 = 61./11.;
   std::cout << "begin iter"<<std::endl;
   for(size_t it=0; it < nt ; it++){
     float temp0 = 3000./x0;
     float temp1 = 1130. - temp0;
     float temp2 = temp1 /x1 ;
     float x2 = 111. - temp2;
     if(verbose){
       cout <<"it: "<< it << "\tx2: "<<x2
            << "\ttemp0: "<<temp0 <<"\ttemp1: "<<temp1<<"\ttemp2: "<<temp2 <<"addr: "<<&temp2<< endl;
     }
     x0 = x1;
     x1 = x2;
   }
   std::cout <<"x["<<nt<<"]="<<x1<<std::endl;
   return x1;
}
```

```
begin iter
it: 0 x2: 5.59016 temp0: 545.455 temp1: 584.545 temp2: 105.410addr: 0x1ffefff858
it: 1 x2: 5.63343 temp0: 540.984 temp1: 589.016 temp2: 105.367addr: 0x1ffefff858
it: 2 x2: 5.67465 temp0: 536.657 temp1: 593.343 temp2: 105.325addr: 0x1ffefff858
it: 3 x2: 5.71333 temp0: 532.535 temp1: 597.465 temp2: 105.287addr: 0x1ffefff858
it: 4 x2: 5.74912 temp0: 528.667 temp1: 601.333 temp2: 105.251addr: 0x1ffefff858
it: 5 x2: 5.78181 temp0: 525.088 temp1: 604.912 temp2: 105.218addr: 0x1ffefff858
it: 6 x2: 5.81131 temp0: 521.819 temp1: 608.181 temp2: 105.189addr: 0x1ffefff858
it: 7 x2: 5.83766 temp0: 518.869 temp1: 611.131 temp2: 105.162addr: 0x1ffefff858
it: 8 x2: 5.86108 temp0: 516.234 temp1: 613.766 temp2: 105.139addr: 0x1ffefff858
it: 9 x2: 5.88354 temp0: 513.904 temp1: 616.096 temp2: 105.116addr: 0x1ffefff858
it: 10 x2: 5.93596 temp0: 511.851 temp1: 618.149 temp2: 105.064addr: 0x1ffefff858
it: 11 x2: 6.53442 temp0: 509.897 temp1: 620.103 temp2: 104.466addr: 0x1ffefff858
x[12]=6.53442
```

nearest

Muller exemple

```
float muller(int nt, bool verbose=false){
   float x0 = 11./2.;
   float x1 = 61./11.;
   std::cout << "begin iter"<<std::endl;
   for(size_t it=0; it < nt ; it++){
     float temp0 = 3000./x0;
     float temp1 = 1130. - temp0;
     float temp2 = temp1 /x1 ;
     float x2 = 111. - temp2;
     if(verbose){
       cout <<"it: "<< it << "\tx2: "<<x2
            << "\ttemp0: "<<temp0 <<"\ttemp1: "<<temp1<<"\ttemp2: "<<temp2 <<"addr: "<<&temp2<< endl;
     }
     x0 = x1;
     x1 = x2;
   }
   std::cout <<"x["<<nt<<"]="<<x1<<std::endl;
   return x1;
}
```

```
begin iter
it: 0 x2: 5.59016 temp0: 545.455 temp1: 584.545 temp2: 105.410addr: 0x1ffefff858
it: 1 x2: 5.63343 temp0: 540.984 temp1: 589.016 temp2: 105.367addr: 0x1ffefff858
it: 2 x2: 5.67465 temp0: 536.657 temp1: 593.343 temp2: 105.325addr: 0x1ffefff858
it: 3 x2: 5.71333 temp0: 532.535 temp1: 597.465 temp2: 105.287addr: 0x1ffefff858
it: 4 x2: 5.74912 temp0: 528.667 temp1: 601.333 temp2: 105.251addr: 0x1ffefff858
it: 5 x2: 5.78181 temp0: 525.088 temp1: 604.912 temp2: 105.218addr: 0x1ffefff858
it: 6 x2: 5.81131 temp0: 521.819 temp1: 608.181 temp2: 105.189addr: 0x1ffefff858
it: 7 x2: 5.83767 temp0: 518.869 temp1: 611.131 temp2: 105.162addr: 0x1ffefff858
it: 8 x2: 5.86108 temp0: 516.234 temp1: 613.766 temp2: 105.139addr: 0x1ffefff858
it: 9 x2: 5.88164 temp0: 513.904 temp1: 616.096 temp2: 105.116addr: 0x1ffefff858
it: 10 x2: 5.97396 temp0: 511.851 temp1: 618.149 temp2: 105.024addr: 0x1ffefff858
it: 11 x2: 6.53442 temp0: 509.797 temp1: 620.203 temp2: 104.466addr: 0x1ffefff858
x[12]=6.53442
```

nearest random

Muller exemple

```
float muller(int nt, bool verbose=false){
   float x0 = 11./2.;
   float x1 = 61./11.;
   std::cout << "begin iter"<<std::endl;
   for(size_t it=0; it < nt ; it++){
     float temp0 = 3000./x0;
     float temp1 = 1130. - temp0;
     float temp2 = temp1 /x1 ;
     float x2 = 111. - temp2;
     if(verbose){
       cout <<"it: "<< it << "\tx2: "<<x2
            << "\ttemp0: "<<temp0 <<"\ttemp1: "<<temp1<<"\ttemp2: "<<temp2 <<"addr: "<<&temp2<< endl;
     }
     x0 = x1;
     x1 = x2;
   }
   std::cout <<"x["<<nt<<"]="<<x1<<std::endl;
   return x1;
}
```

```
begin iter
it: 0 x2: 5.59016 temp0: 545.455 temp1: 584.545 temp2: 105.410addr: 0x1ffefff858
it: 1 x2: 5.63343 temp0: 540.984 temp1: 589.016 temp2: 105.367addr: 0x1ffefff858
it: 2 x2: 5.67465 temp0: 536.657 temp1: 593.343 temp2: 105.325addr: 0x1ffefff858
it: 3 x2: 5.71333 temp0: 532.535 temp1: 597.465 temp2: 105.287addr: 0x1ffefff858
it: 4 x2: 5.74912 temp0: 528.667 temp1: 601.333 temp2: 105.251addr: 0x1ffefff858
it: 5 x2: 5.78181 temp0: 525.088 temp1: 604.912 temp2: 105.218addr: 0x1ffefff858
it: 6 x2: 5.81131 temp0: 521.819 temp1: 608.181 temp2: 105.189addr: 0x1ffefff858
it: 7 x2: 5.83766 temp0: 518.869 temp1: 611.131 temp2: 105.162addr: 0x1ffefff858
it: 8 x2: 5.86108 temp0: 516.234 temp1: 613.766 temp2: 105.139addr: 0x1ffefff858
it: 9 x2: 5.88154 temp0: 513.904 temp1: 616.096 temp2: 105.116addr: 0x1ffefff858
it: 10 x2: 5.89996 temp0: 511.851 temp1: 618.149 temp2: 105.099addr: 0x1ffefff858
it: 11 x2: 6.53442 temp0: 509.797 temp1: 620.203 temp2: 104.466addr: 0x1ffefff858
x[12]=6.53442
```

nearest random random

Muller exemple

```
float muller(int nt, bool verbose=false){
   float x0 = 11./2.;
   float x1 = 61./11.;
   std::cout << "begin iter"<<std::endl;
   for(size_t it=0; it < nt ; it++){
     float temp0 = 3000./x0;
     float temp1 = 1130. - temp0;
     float temp2 = temp1 /x1 ;
     float x2 = 111. - temp2;
     if(verbose){
       cout <<"it: "<< it << "\tx2: "<<x2
            << "\ttemp0: "<<temp0 <<"\ttemp1: "<<temp1<<"\ttemp2: "<<temp2 <<"addr: "<<&temp2<< endl;
     }
     x0 = x1;
     x1 = x2;
   }
   std::cout <<"x["<<nt<<"]="<<x1<<std::endl;
   return x1;
}
```

```
begin iter
it: 0 x2: 5.59016 temp0: 545.455 temp1: 584.545 temp2: 105.410addr: 0x1ffefff858
it: 1 x2: 5.63343 temp0: 540.984 temp1: 589.016 temp2: 105.367addr: 0x1ffefff858
it: 2 x2: 5.67465 temp0: 536.657 temp1: 593.343 temp2: 105.325addr: 0x1ffefff858
it: 3 x2: 5.71333 temp0: 532.535 temp1: 597.465 temp2: 105.287addr: 0x1ffefff858
it: 4 x2: 5.74912 temp0: 528.667 temp1: 601.333 temp2: 105.251addr: 0x1ffefff858
it: 5 x2: 5.78181 temp0: 525.088 temp1: 604.912 temp2: 105.218addr: 0x1ffefff858
it: 6 x2: 5.81131 temp0: 521.819 temp1: 608.181 temp2: 105.189addr: 0x1ffefff858
it: 7 x2: 5.83766 temp0: 518.869 temp1: 611.131 temp2: 105.162addr: 0x1ffefff858
it: 8 x2: 5.86108 temp0: 516.234 temp1: 613.766 temp2: 105.139addr: 0x1ffefff858
it: 9 x2: 5.88154 temp0: 513.904 temp1: 616.096 temp2: 105.116addr: 0x1ffefff858
it: 10 x2: 5.89996 temp0: 511.851 temp1: 618.149 temp2: 105.099addr: 0x1ffefff858
it: 11 x2: 6.53442 temp0: 509.897 temp1: 620.103 temp2: 104.466addr: 0x1ffefff858
x[12]=6.53442
```

nearest random random random

Muller exemple

```
float muller(int nt, bool verbose=false){
   float x0 = 11./2.;
   float x1 = 61./11.;
   std::cout << "begin iter"<<std::endl;
   for(size_t it=0; it < nt ; it++){
     float temp0 = 3000./x0;
     float temp1 = 1130. - temp0;
     float temp2 = temp1 /x1 ;
     float x2 = 111. - temp2;
     if(verbose){
       cout <<"it: "<< it << "\tx2: "<<x2
            << "\ttemp0: "<<temp0 <<"\ttemp1: "<<temp1<<"\ttemp2: "<<temp2 <<"addr: "<<&temp2<< endl;
     }
     x0 = x1;
     x1 = x2;
   }
   std::cout <<"x["<<nt<<"]="<<x1<<std::endl;
   return x1;
}
```

```
begin iter
it: 0 x2: 5.59016 temp0: 545.455 temp1: 584.545 temp2: 105.410addr: 0x1ffefff858
it: 1 x2: 5.63343 temp0: 540.984 temp1: 589.016 temp2: 105.367addr: 0x1ffefff858
it: 2 x2: 5.67465 temp0: 536.657 temp1: 593.343 temp2: 105.325addr: 0x1ffefff858
it: 3 x2: 5.71333 temp0: 532.535 temp1: 597.465 temp2: 105.287addr: 0x1ffefff858
it: 4 x2: 5.74912 temp0: 528.667 temp1: 601.333 temp2: 105.251addr: 0x1ffefff858
it: 5 x2: 5.78181 temp0: 525.088 temp1: 604.912 temp2: 105.218addr: 0x1ffefff858
it: 6 x2: 5.81131 temp0: 521.819 temp1: 608.181 temp2: 105.189addr: 0x1ffefff858
it: 7 x2: 5.83766 temp0: 518.869 temp1: 611.131 temp2: 105.162addr: 0x1ffefff858
it: 8 x2: 5.86108 temp0: 516.234 temp1: 613.766 temp2: 105.139addr: 0x1ffefff858
it: 9 x2: 5.88184 temp0: 513.904 temp1: 616.096 temp2: 105.118addr: 0x1ffefff858
it: 10 x2: 5.89996 temp0: 511.851 temp1: 618.149 temp2: 105.100addr: 0x1ffefff858
it: 11 x2: 5.91544 temp0: 509.997 temp1: 620.003 temp2: 104.085addr: 0x1ffefff858
x[12]=5.91544
```

nearest random random random random

Localisation temporelle intrusive

```cpp
#include "valgrind/libverrouTask.h"
float muller(int nt, bool verbose=false){
   float x0 = 11./2.;
   float x1 = 61./11.;
   std::cout << "begin iter"<<std::endl;
   for(size_t it=0; it < nt ; it++){
     verrou_task("muller_iter", it);
     float temp0 = 3000./x0;
     float temp1 = 1130. - temp0;
     float temp2 = temp1 /x1 ;
     float x2 = 111. - temp2;
     if(verbose){ ... }
     x0 = x1;
     x1 = x2;
   }
   std::cout <<"x["<<nt<<"]="<<x1<<std::endl;
   return x1;
}
int main(int argc, char** argv){
    verrou_task_init();
    float muller(12, verboseFromArgs(argc,argv));
    verrou_task_finalize();
}
```

- Compilation avec `-lverrouTask`, mais pourrait être implémenté en interne du frontend verrou via des client-request.
- Automatisable en python :

```
#!/bin/bash
DIR=$1
valgrind --tool=verrou --rounding-mode=downward \
         trace_verrou_task.py ./Muller.py > $DIR/res.dat
```

Delta-debug: `verrou_dd_task`

Delta-debug search:

```
verrou_dd_task --nruns=5 ddRun.sh extractOrCmp.py

    ddmin0 (                ddmin1 (
    muller_iter:0):         muller_iter:1):
```

runScript: `ddRun.sh`

```
#!/bin/bash
PREFIX="valgrind --tool=verrou
                 --rounding-mode=random"
$PREFIX ./muller-task -v > $1/res.dat
#warning: ./muller => ./muller-task
```

cmpScript: `extractOrCmp.py`

```
#!/usr/bin/python3
import sys
from os.path import join
def extractValue(rep):
    for line in (open(join(rep,"res.dat")).readlines()):
        if line.startswith("x[12]="):
            return float(line.partition("=")[2])

if __name__=="__main__":
    if len(sys.argv)==2:
        print(extractValue(sys.argv[1]))
    if len(sys.argv)==3:
        valueRef=extractValue(sys.argv[1])
        value=extractValue(sys.argv[2])
        relDiff=abs((value-valueRef)/valueRef)
        if relDiff < 1.e-2: sys.exit(0) #OK
        else: sys.exit(1)               #KO
```

Remarques:

- Localisation quasi-temporelle;
- Robuste au contexte multi-processus.

Limitations :

- Besoin de recompiler ;
- Enchaînement depuis des résultats de `verrou_dd_line` coûteux.

Localisation temporelle sans recompilation

Nouvel espace de recherche: wildcarded IO (automatique grâce à des regexp ou défini manuellement)

```
begin iter
it: 0 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 1 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 2 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 3 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 4 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 5 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 6 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 7 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 8 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 9 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 10 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 11 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
x[12]=*
```

```
verrou_dd_stdout --nruns=5 ddRun.sh extractOrCmp.py
```

```
ddmin0 (
|begin iter|):
...
ddmin1 (
|it: 0 x2: * temp0: * temp1: * temp2: *addr: 0x??????????|):
```

- La sortie standard (ou un fichier) *match* sans contexte temporel;
- L'utilisateur doit faire attention à la bufferisation;
- Les lignes vides peuvent être ignorée;
- La sortie standard peut être modifiée par un script.

IOMatch format

The option `-IOmatch-clr=IOMATCH_FILE` (or environnement variable `VERROU_IOMATCH_CLR`) enables interaction between stdout (or file) and verrou.

- setup keys: `verbose: LEVEL`, `ignore-empty-line:` , `filter_line_exec: CMD`, `dump-stdout: [FILENAME]`, `dump-filtered-stdout: [FILENAME]`
- definition of match section:
  - `[break] bmatch: PATTERN`, `apply: ACTION`, `post-apply: ACTION`.
  - `[continue] cmatch: PATTERN`, `apply: ACTION`.
- definition of specific action: `default: ACTION`, `init: ACTION`, `post_init: ACTION`.

The available actions are:

- `start`, `stop`, `start_soft`, `stop_soft`,
- `display_counter`, `nb_instr`, `reset_counter`,
- `panic`, `exit`, `dump_cover`,
- `nop`,`default`, `init`, `post-init`.

Autres éléments d'implémentation

- modification de l'interface pour le delta-debug stochastique;
  - besoin de conversions entre l'espace de recherche/configuration et les données lues/générées par verrou ;
  - besoin de gérer des options spécifiques pour chaque outil `verrou_dd_*` .
- refonte des clients request stop/start.
  - hard stop/start:
    - choix du mode d'arrondi et de l'instrumentation fais lors de l'instrumentation
    - nécessité de nettoyer le cache d'instrumentation
    - performance du code non instrumenté proche de celle de `-tool=none`
  - soft stop/start
    - choix mode d'arrondi fait lors de l'instrumentation
    - instrumentation par un wrapper qui sélectionne dynamiquement entre le backend nearest et backend verrou (appel spécifique au mode d'arrondi).
    - génération de code conséquente.
    - performance du code non instrumenté proche de celle de `-rounding-mode=nearest` (un test en plus par instruction)
  - compatibilité entre client request soft et hard:
    - intersection de l'instrumentation soft et hard;
    - privilégier soft pour des changements très fréquents;
    - privilégier hard pour des changements peu fréquents;
    - si un code utilise déjà des clients request stop/start, `verrou_dd_task` et `verrou_dd_stdout` doivent utiliser l'autre mode pour éviter des conflits.

Exemple d'usage avancée de verrou_dd_stdout

- Algorithme itératif qui corrige ses erreurs au prix de nouvelles itérations

```
verrou_dd_stdout --IOmatch-header=iomatch.header ddRun.sh extractOrCmp.py
```

Fichier iomatch.header:
```
cmatch: it: 15=*
apply: exit
```

- Message de debug intempestif;

```
begin iter
debug
it: 0 x2: 5.59016 temp0: 545.455 temp1: 584.545 temp2: 105.41addr: 0x7fffe0b19328
debug
it: 1 x2: 5.63343 temp0: 540.984 temp1: 589.016 temp2: 105.367addr: 0x7fffe0b19328
debug
...
```

```
verrou_dd_stdout --filter-cmd="/usr/bin/sed -u s/debug.*//" ddRun.sh extractOrCmp.py
```

- le script est exécuté une fois par exécution du binaire (et non par ligne);
- le filtre peut être exécuté en python (en faisant attention à la bufferisation).
  - Possibilité de réintroduire du contexte (utile pour les itérations imbriquées);
  - Possibilité de fusionner des itérations.

Conclusion

- La localisation (quasi)-temporelle (intrusive et non intrusive) sont fonctionnelles ;
- La localisation en python est fonctionnelle ;
- A part le changement de format de sortie, la recompilation est évitable.

Perspectives

- Améliorer l'ergonomie:
  - `post_verrou_dd` pour `verrou_dd_task` et `verrou_dd_stdout`;
  - message d'erreurs pour les filtres invalides;
  - avertissement pour les problèmes de bufferisation;
  - déactivation de plus de bufferisation;
  - verrou_task par client-request.
- Localisation par backtrace;
- Peut-on accélerer les recherches via des mécanismes arrêt/reprise?
- Amélioration des performances du frontend (partie instrumentée et non instrumentée).
- Localisation amplification d'erreur (branchements et annulations)

Available on github:
`http://github.com/edf-hpc/verrou`

Documentation:
`http://edf-hpc.github.io/verrou/vr-manual.html`