

Espace d'adressage global partitionné

Félix-Antoine Ouellet

Université de Sherbrooke

2 octobre 2014

- 1 Motivation
- 2 Espace d'adressage global partitionné
- 3 Implémentation
- 4 Conclusion

Plan

- 1 Motivation
 - État présent du matériel
 - État présent du logiciel
- 2 Espace d'adressage global partitionné
- 3 Implémentation
- 4 Conclusion

Explosion de parallélisme

Appareils courants

- Processeurs vectoriels
- Processeurs multi-coeurs
- Accélérateurs

Explosion de parallélisme

Superordinateurs

- Aux portes de l'*exascale computing*

Explosion de parallélisme

Superordinateurs

- Aux portes de l'*exascale computing*
- 10^{18} opérations en virgule flottante par seconde

Explosion de parallélisme

Superordinateurs

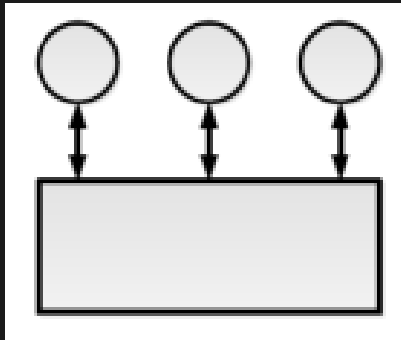
- Aux portes de l'*exascale computing*
- 10^{18} opérations en virgule flottante par seconde
- Potentiellement 1 milliard de *threads* à gérer simultanément

Explosion de parallélisme

Superordinateurs

- Aux portes de l'*exascale computing*
- 10^{18} opérations en virgule flottante par seconde
- Potentiellement 1 milliard de *threads* à gérer simultanément
- Pas nécessairement utiliser par des informaticiens

Programmation parallèle avec mémoire partagée



Programmation parallèle avec mémoire partagée

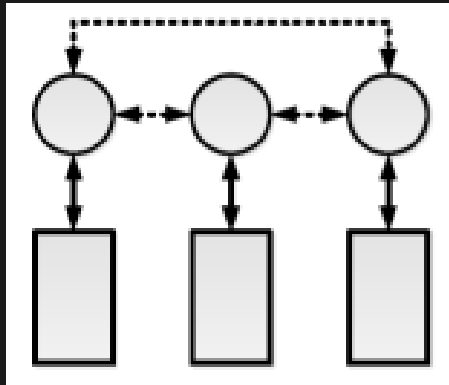
Avantages:

- Raisonnement plus facile
- Unique espace d'adressage

Inconvénients:

- Conditions de course
- N'échelonne pas bien

Programmation parallèle avec mémoire distribuée



Programmation parallèle avec mémoire distribué

Avantages:

- S'échelonne bien
- Pas de conditions de course

Inconvénients:

- Doit penser à la distribution des données
- Performance lié au réseau

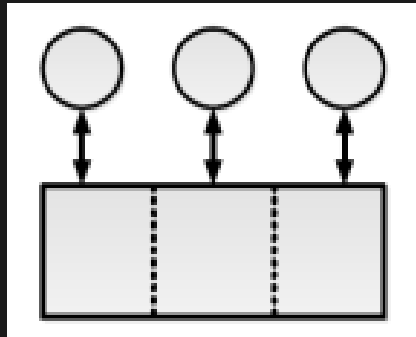
Programmation parallèle

- Des abstractions de trop bas niveau nuisent à la productivité
- Des abstractions de trop haut niveau nuisent à la performance

Plan

- 1 Motivation
- 2 Espace d'adressage global partitionné
 - Programmation à haut niveau
 - Localité
 - Tâches
- 3 Implémentation
- 4 Conclusion

Espace d'adressage global partitionné



Programmation à haut niveau

- L'espace d'adressage apparaît unifié
- Peu de notions de parallélisme
- Le compilateur s'occupe de faire marcher le tout

Programmation à haut niveau

Exemple

```
Vector<int> Add(Vector<int> A,  
                Vector<int> B) {  
    Vector<int> C;  
    parfor(int i = 0; i < 100000000; ++i) {  
        C[i] = A[i] + B[i];  
    }  
  
    return C;  
}
```

Programmation à haut niveau

- Pas nécessairement la version optimale qui sera produite
- Par contre, il s'agit de la version qui maximise la productivité

Localité

- Faire la distinction entre ce qui est global et ce qui est local
- Contrôler la distribution des données

Localité

- Faire la distinction entre ce qui est global et ce qui est local
- Contrôler la distribution des données
 - Profiter des hiérarchies mémoire modernes
 - Profiter de divers types de processeurs

Localité

Code

```
global Vector<int> Vec;
global int TotalSum
/*...*/
for(int i = 0; i < nbLocales(); ++i) {
    do on Locale[i] {
        int PartialSum = 0;
        for(int j = 0; j < ; ++j)
            PartialSum += Vec[j];
        TotalSum += PartialSum;
    }
}
```

Tâches

Les bases

- Ensemble d'instructions à exécuter
- Repose sur un *thread pool*
- Possiblement lié à une localité

Tâches

Les bases

```
void Func() {  
    int Res = CalculComplexe(this_locale.ID);  
    print("Resultat:", Res);  
}  
/*...*/  
for(int j = 0; j < nbLocales(); ++j) {  
    do on Locale[j] {  
        Func();  
    }  
}
```

Tâches

Fonctionnalités avancées

- Exploiter le calcul asynchrone
- Créer dynamiquement des tâches
- Invoquer des tâches situées sur une autre localité

Tâches

Fonctionnalités avancées

```
void Func() {  
    int Res = 0;  
    parfor(int i = 0; i < 100; ++i)  
        Res += CalculComplexe(this_locale.ID);  
    println("Locale:", this_locale.ID);  
    println("Resultat:", Res);  
}  
/*...*/  
do on Locale[3] {  
    async Func();  
}  
println("Locale:", this_locale.ID);
```

Plan

- 1 Motivation
- 2 Espace d'adressage global partitionné
- 3 Implémentation**
- 4 Conclusion

DARPA HPCS

- *High Productivity Computing Systems*
- But: Produire des systèmes informatiques hautement productif pour l'industrie et la sécurité nationale

Chapel

Présentation

- Réponse de Cray au projet HPCS
- Inspiré de langage comme C, C++, C#, Java, Fortran, HPF

Chapel

Exemple - Calcul de PI

```
const numRect = 10000000;  
const D : domain(1) = 1..numRect;  
const width = 2.0 / numRect;  
const baseX = -1 - width/2;  
proc rectangleArea(i : int) {  
    const x = baseX + i*width;  
    return width * sqrt(1.0 - x*x);  
}  
var halfPI : real;  
for i in D {  
    halfPI += rectangleArea(i);  
}  
writeln("Result:", 2*halfPI);
```

Plan

- 1 Motivation
- 2 Espace d'adressage global partitionné
- 3 Implémentation
- 4 Conclusion

Conclusion

- Les nouveaux défis en calcul de haute performance demande de nouvelles réponses

Conclusion

- Les nouveaux défis en calcul de haute performance demande de nouvelles réponses
- L'avènement de l'ère multi-coeurs va forcer les langages de programmation à évoluer