Décision de finales d'échecs

Mathis Caristan & Alexandre Fernandez

UPMC

28 Mars 2017

Table des matières

- **Idée :** Paralléliser la boucle `for` dans `evaluate`
- Profondeur max : 2 $\longrightarrow$ ~30 noeuds
- Deux choix d'implémentations possibles

  a°) `parallel for` (plus simple)
  b°) `task` (plus efficace)

```
#pragma omp parallel firstprivate(T,result)
#pragma omp single
evaluate(T, result, 0);
```

```
if(prof < OMP_MAX_PROF) {
#pragma omp parallel for
    for (int i=0 ; i<n_moves ; i++) {
        fct_for(T, moves[i], &child[i], &child_result[i], prof, result);
    }
}
// Sinon appel a fct_for sans #pragma omp parallel for

void fct_for(tree_t *T, move_t m, tree_t *child, result_t *child_res,
        int prof, result_t *cur_res) {
    play_move(T, m, child);

    evaluate(child, child_res, prof);

    int child_score = -child_res->score;

#pragma omp critical(CHILD)
    { // BLOCK OMP : critical
        if (child_score > cur_res->score) {
            cur_res->score = child_score;
            cur_res->best_move = m;
            cur_res->pv_length = child_res->pv_length+1;
            for (int j=0 ; j<child_res->pv_length ; j++)
                cur_res->PV[j+1] = child_res->PV[j];
            cur_res->PV[0] = m;
        }
    } // BLOCK OMP : critical
}
```

- Bloc `critical` pour protéger les accès concurents à cur_res
- Factorisation du code dans la fonction `fct_for`

- `play_move` et `evaluate` sont lancés dans des tâches
- Barrière de synchornisation à la fin du `for`
- La recherche du maximum est effectuée sur un tableau pour améliorer les performances

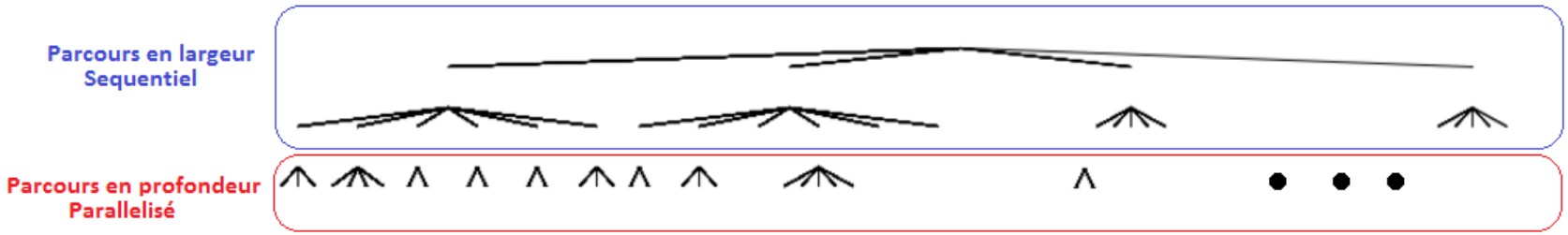
```
if(prof < OMP_MAX_PROF) {
    int child_score[n_moves];
    tree_t child[n_moves];
    result_t child_result[n_moves];
    for (int i=0 ; i<n_moves ; i++) {

#pragma omp task firstprivate(i) shared(child_result, child_score, child)
        {
            play_move(T, moves[i], &child[i]);

            evaluate(&child[i], &child_result[i], prof+1);

            child_score[i] = -child_result[i].score;
        }
    }
#pragma omp taskwait
    int max = 0;
    for(int i=0; i<n_moves; i++) {
        if(child_score[i] > child_score[max]) max = i;
    }
    if (child_score[max] > result->score) {
        result->score = child_score[max];
        result->best_move = moves[max];
        result->pv_length = child_result[max].pv_length + 1;
        for(int j = 0; j < child_result[max].pv_length; j++)
            result->PV[j+1] = child_result[max].PV[j];
        result->PV[0] = moves[max];
    }
}
```

- **Idée :** Atteindre une profondeur suffisante pour avoir une multiplicité de tâches intéressante.
- Pré-calcul (séquentiel) consistant à un parcours en largeur de l'arbre.
- Équilibrage de charge dynamique
- Réglages nécessaires selon les conditions d'utilisation



```
for (int i=beg ; i<bound ; i++) {
    node_searched++;
    tree_t *tmpTree = preEvalTrees[i].tree;
    result_t *tmpResult = preEvalTrees[i].result;

    tmpResult->score = -MAX_SCORE-1;
    tmpResult->pv_length = 0;
    // (...) tests
    compute_attack_squares(tmpTree);
    n_moves = generate_legal_moves(tmpTree, moves);
    // (...) test
    beg = sizeTree;
    sizeTree += n_moves;
    sum += n_moves;
    preEvalTrees = realloc(preEvalTrees, sizeTree*sizeof(recTree_t))
    for (int j=beg ; j<sizeTree ; j++) {
          preEvalTrees[j].parentId = i;
          preEvalTrees[j].move = moves[j-beg];
          preEvalTrees[j].tree = malloc(sizeof(tree_t));
          preEvalTrees[j].result = malloc(sizeof(result_t));
          play_move(preEvalTrees[i].tree, moves[j-beg],
preEvalTrees[j].tree);
    }
}
```

- La structure d'arbre des données est peu utilisée
- On remonte l'arbre au lieu de le descendre
- On ne garde une référence que vers le parent, pas les enfants

- **Idée :** Reprendre les principes de MPI et OpenMP en les combinant
- Utilisation d'OpenMP pour paralléliser le pré-calcul
- Pas encore implémenté

Parrallel for : seq time = ~20.5 s
Task : seq time = ~ 15.5 s

Efficacités moyennes :

Tests avec 4 coeurs	Taux Max_Prof=1	Taux Max_Prof=2	Taux Max_Prof=3	Taux Max_Prof=4
OpenMP //for	0.54	0.54	0.54	0.54
OpenMP tasks	0.62	0.73	0.78	0.81

- On note une dépendance de plusieurs paramètres
- **Objectif :** Chercher à determiner la dépendance vis-à-vis des paramètres (fit?)