組込みシステム向けFRP言語の実行モデルの並列化

櫻井義孝・渡部卓雄

東京工業大学

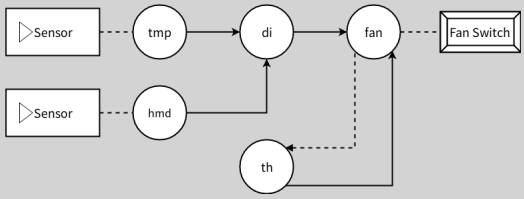
概要

- 本研究の目的は小規模組込みシステム向け FRP 言語 Emfrp の応答性向上である．
- 既存の小規模組込みシステム向け FRP 言語である Emfrp の実行モデルはシングルスレッドにしか対応していないため，マルチコア CPU の上で動作させたときに十分に計算資源を活用できない．
- 本研究では応答性向上のために静的スケジューリングを用いて Emfrp の実行モデルを並列化した純粋 FRP 言語 XFRP-Core を開発した．

Emfrp [Sawada2016]

小規模組込みシステム向け FRP 言語 Emfrp

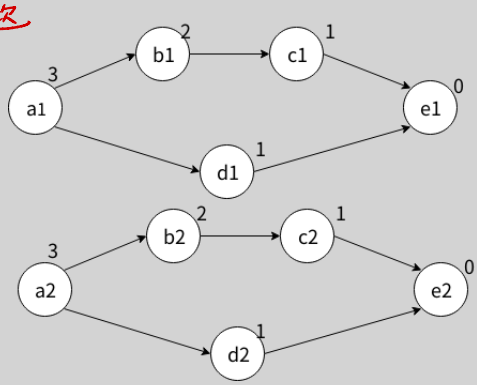
```
module FanController     # 不快指数からファンの
in  tmp: Double,         # オンオフを決定するアプリケーション
    hmd: Double
out di:  Double,
    fan: Bool
use Std

# discomfort index
node di = 0.81*tmp+0.01*hmd*(0.99*tmp-14.3)+46.3

# fan switch
node init[False] fan = di >= th

# threshold
node th = 75.0 + (if fan@last then -0.5 else 0.5)
```



- Emfrp のプログラムは，時変値をノードとし依存関係を辺とした有向非巡回グラフ (DAG) を構成する．
- 現在の Emfrp の処理系ではノードの更新は逐次的に行われる．左に挙げた例の場合，DAG をトポロジカルソートした tmp→hmd→di→th→fan の順に更新が行われる．この更新 (サイクル) を繰り返すことでリアクティブな動作を実現している．
- 1 サイクルにかかる時間がシステムの応答性を決める．本研究では，マルチコアシステムでの応答性向上を目的とした並列化方式を提案する．実行環境として汎用的な OS だけでなく，OS のない環境や FreeRTOS 程度の小規模 RTOS を考える．
- 本研究の想定環境には OS によるスケジューラによる適切なスケジューリングが期待できない環境も含まれている．そのため，実行モデルを並列化する際に実行モデルは OS のスケジューラに依存しないで動作する必要がある．
- 本研究では，各サイクルの計算時間を短縮し，可能であれば応答時間を予測できるような静的スケジューリング方式を提案する．

並列化アルゴリズム

本研究が提案するアルゴリズムはコンパイル時に静的にスケジューリングする．ただし，コンパイル時に使用するスレッド数は決定されていることを前提とする．下の図は実験で用いたアプリケーションを表現するグラフの一部であり，これを例とする．提案する並列化アルゴリズムは各ノードから最も遠い到達可能な sink ノードへの距離を基準にスケジューリングを行う．ここで sink ノードとはグラフで出自数が 0 のノードであり，右の例ではノード e_1, e_2 が相当する．グラフのノードの右上の数字が最も遠い到達可能な sink ノードへの距離である．並列化はこの最も遠い到達可能な sink ノードへの距離が等しいノードを並列に実行することで実現される．例えば，右のグラフのノード a_1, a_2 は等しく距離 3 なので同時に実行可能，ノード c_1, d_1, c_2, d_2 は等しく距離 1 なので同時に実行可能となる．同時に実行可能なノードの数が N 個のとき，M スレッドで処理を行うならば 1 スレッドあたり N/M 個のノードを更新すればよい．例えば 2 スレッドで右図を処理するならば，最長距離 1 のノードを更新する際は 1 スレッドあたり 2 つのノードを更新すれば良いのでスレッド 1 が c_1, d_1 を，スレッド 2 が c_2, d_2 を更新するというように更新する時変値を割り当てれば良い．



実験

提案する並列化アルゴリズムの性能を評価するために，XFRP-Core 上で提案する実行モデルと Emfrp の実行モデルを実装し，10^5 サイクルにかかる時間を測定した．実験は，Core i7-975 上の Ubuntu12.04 で行った．並列化には Pthread を使用し，同期には Pthread ライブラリで実装されているバリア同期を使用した．実験では Emfrp の実行モデルは 1 スレッドで実行し，XFRP-Core は 2 スレッドと 4 スレッドでそれぞれ 10 回計測し，その平均時間を実験結果とした．実験対象のアプリケーションとして，LifeGame と熱拡散シュミレータを XFRP 上で実装した．アプリケーションで使用される時変値の数はそれぞれ 5×10^5 個と 3×10^5 個となった．実験結果は以下の表のようになった．

アプリケーション	Emfrp	XFRP-Core(2)	XFRP-Core(4)
LifeGame	203.61(sec)	108.04(sec)	64.05(sec)
熱拡散シュミレータ	65.70(sec)	37.09(sec)	25.88(sec)

LifeGame については，XFRP-Core の実行モデルで 2 スレッドと 4 スレッドを使用したときの実行時間は Emfrp の実行モデルの場合と比べてそれぞれ 53.0% と 31%になった．また，熱拡散シュミレータに対してはそれぞれ 56.5%,39%の実行時間になった．