■ Ubuntu の環境設定

ThinkPad X60 にインストールしている Ubuntu ですが,
やはり，Gnome な環境は操作が面倒なので，ウィンドウマネージャを fvwm2 に変更.
gnome による設定 (？) に頼っていたキーボードとかマウスの設定を手で書いて,
あと，gnome-panel とかネットワーク設定とかの便利ツールだけ
起動するように変更．快適♪

gnome-panel とかアプレット

.fvwm2rc の起動コマンドで実行することに.

```
AddToFunc "InitFunction" "I" Module FvwmWinList
AddToFunc "InitFunction" "I" Exec gnome-panel &
AddToFunc "InitFunction" "I" Exec gnome-terminal --geometry=+10+10 &
AddToFunc "InitFunction" "I" Exec nm-applet &
AddToFunc "InitFunction" "I" Exec oclock -fg black -bd black -transparent -geometry 100x100-0-0 &
AddToFunc "InitFunction" "I" Exec gnome-keyring-daemon &
AddToFunc "InitFunction" "I" Exec skype &
```

こんな感じ．もっと増えるような気もするけど.

■オンチップ制御並列プロセッサ MUSCAT の提案

```
@article{ 鳥居淳 :19980615,
author=" 鳥居 淳 and 近藤 真己 and 本村 真人 and 池野 晃久 and 小長谷 明彦 and 西 直樹 ",
title=" オンチップ制御並列プロセッサ MUSCAT の提案 (< 特集 > 並列処理 )",
journal=" 情報処理学会論文誌 ",
ISSN="03875806",
publisher=" 社団法人情報処理学会 ",
year="19980615",
volume="39",
number="6",
pages="1622-1631",
URL="http://ci.nii.ac.jp/naid/110002722175/",
DOI="",
}
```

・制御並列アーキテクチャ
- 命令レベル並列処理の理論的な性能限界
- 広範囲な領域における性能向上と自動並列化コンパイラの実現
- PE 間の距離が短く交信コストが低いという利点を活かした制御並列処理

表 1 より

	並列性抽出	データ転送	メモリ正依存	メモリ逆依存
MUSCAT	コンパイラ	フォーク時レジスタ継承	同期 / データ SP	自動解消
Multiscalar	コンパイラ	任意時レジスタ継承	データ SP	ARB 自動解消
SPSM	コンパイラ	フォーク時 / 後にマージ	データ SP	自動解消

Superthread	コンパイラ	メモリバッファ経由	同期	メモリバッファ
PEWs	ハードウェア	任意時レジスタキュー経由	データ SP	キャッシュ内自動解消
TLS	コンパイラ	キャッシュ経由	データ SP	キャッシュ内自動解消
Multi-superscalar pipeline	ハードウェア	親レジスタ参照可	同期	ハードウェア
OCHA-Pro	ハードウェア	キャッシュ経由	データ SP	SAB 自動解消
SKY	コンパイラ	任意時レジスタ継承	並列化しない	並列化しない

MUSCAT の特色

- フォーク 1 回モデル
  - FORK/TERM をコードに付加
  - 同時存在スレッド数は PE 数 +1 以内
  - 各スレッドの世代を定義できる
  - デッドロックフリー
  - フォーク先は隣接 PE に限定可能
  - 特別な同期機構が不要
- フォーク時レジスタ継承
  - FORK 命令実行時点のレジスタセットを後続スレッドで継承
  - 先行 / 後続スレッド間のデータ転送にはレジスタ間直接転送命令を用いない
- スレッドスペキューレション
  - 実行の取消しが H/W 上可能な範囲で
  - SPFORK(生成)/THFIX(確定)/THABORT(破棄)
- 2 種類のメモリデータ依存性制御
  - フォーク後にストアするアドレスが解析可能であれば BLOCK 宣言でロック (RELEASE で開放)
  - 解析不可能な場合 (1): FORK の発行を遅らせる
  - 解析不可能な場合 (2): 投機的なロード / ストア (= データスペキュレーション)
  - データスペキュレーション失敗は out-of-order スーパスカラの状態復帰機構
  - 逆依存はストアバッファを用いて H/W で解消
- 制御並列処理向きの同期命令
  - 逐次順序関係の維持による暗黙の同期
  - PWAIT(先行スレッドの終了をまつ)/CWAIT(スレッドの実行確定をまつ)

コード生成

- フォーク生成 : 投棄実行の少い実行とレジスタ継承の活用のトレードオフ
- ループ間は継承すべきレジスタ値確定後にフォーク.
  - 静的に決定できない場合 SPFORK 命令
- データ依存
- FORK1 回の保証

評価

・4 命令 / サイクル．スレッド生成は隣接 PE に対し 1 サイクル．
・idct/p_bloc(SPEC gcc の propagate_block)/compress(SPEC)/eqntott(SPEC)
・自動並列化可能な範囲で書きかえたプログラムを使用

スーパスカラとの比較

・コード生成手法により，実行が確実 / 確率が高いスレッドが優先される
　・ 分岐予測ペナルティが課せられない
・命令ウィンドウの分割によるプリフェッチ効果
・スーパスカラは命令ウィンドウサイズがループの 1 反復に限られる
　・アンローリングしても使用可能な論理レジスタ本数による制限をうける

課題

・PE 数が増えた場合，キャッシュ容量 / 理想数の不足からスラッシングが生じる