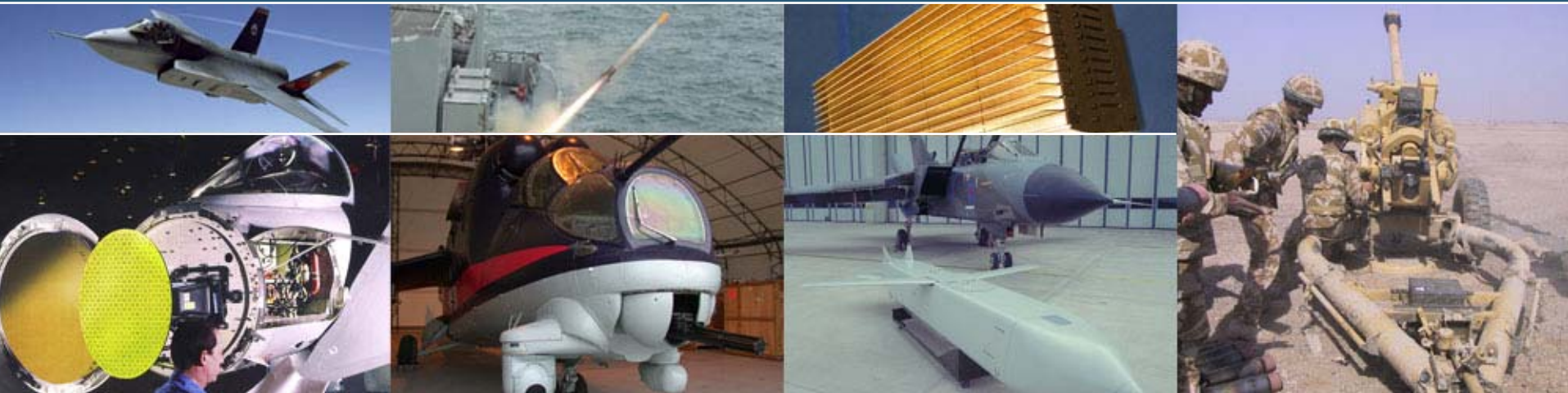


Use of Advanced Gedaе Features on CAPTOR Tranche 2

Dr Marcus Alphey

CAPTOR Tranche 2 Systems & Software IPT
BAE SYSTEMS Avionics, Sensor Systems Division



Overview

Introduction

Gedae and CAPTOR

Advanced Gedae Features

Improvements in Modelling Capability

Improvements in Memory Use

Improvements in Application Executable Generation

Conclusions



Introduction

- The CAPTOR programme is developing the multi-mode nose radar for the Eurofighter Typhoon
 - Tranche 1 (T1): baseline radar, now in production and being flown
 - Tranche 2 (T2): re-development to eliminate hardware obsolescence issues
- T2 Signal Processing Software (SPS) consists of:
 - centralised framework controller
 - multiple round-robin processing engines, each capable of running any mode
- Gedae selected for T2 re-development
 - Facilitates porting to COTS hardware
 - Excellent support for distributed, multi-processor target mapping and optimisation
 - Ideal for large, complicated signal processing applications
 - T2 SPS structure well-suited to Gedae's hierarchical, modular design philosophy

Gedae and CAPTOR

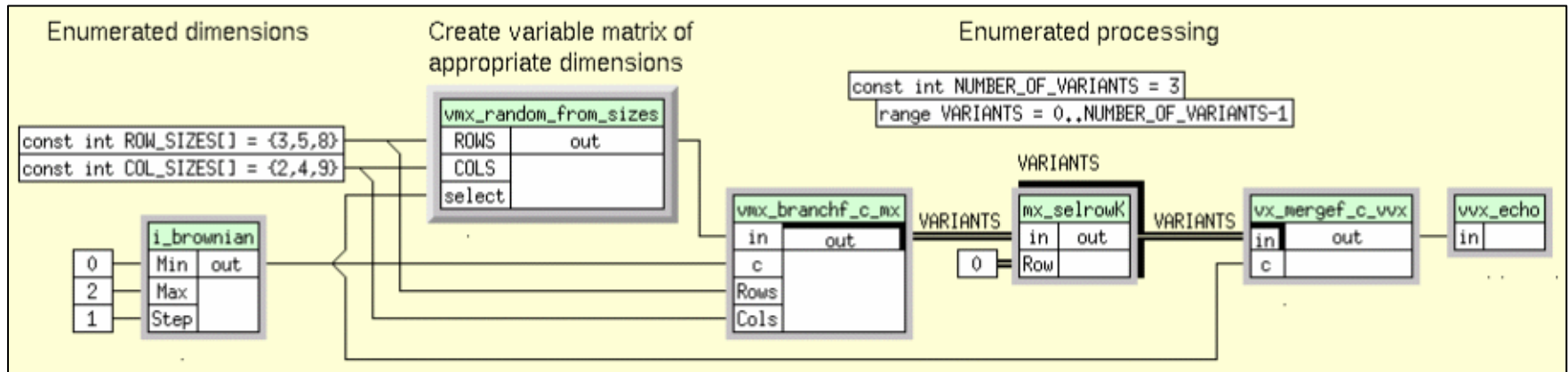
- CAPTOR Tranche 2 implementational and modelling constraints
 - Project requirements define in great detail the resources available:
 - Amount of non-volatile memory from which application must launch
 - Time in which this must be achieved
 - Memory available to each processor while running application
 - Required throughput of data for each mode
 - Clear, maintainable and efficient design
 - Generic, re-useable, conceptually simple components
- At point of adoption for the development of T2:
 - CAPTOR was largest project ever to be undertaken in Gedae
 - Gedae lacked features to support resource optimisation issues
 - Modelling issues existed that had never been faced by Gedae
 - However, clear potential for extending Gedae to resolve this
 - Only consider modifications of general use to wider Gedae community
 - Prioritised new features, agreed timescales and release schedule

Advanced Gedae Features

- Generically useful, but technically advanced
 - Require good understanding of scheduling, memory management, etc.
 - Of greatest advantage in large or complicated applications
- Made development of T2 SPS considerably easier
- Three main categories:
 - Modelling Improvements: how application can be described
 - Operational Memory Improvements: amount of required schedule memory
 - Application Generation Improvements: content and size of executable

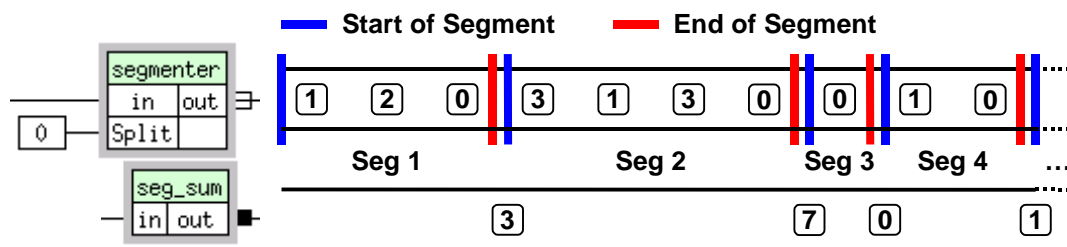
Improvements in Modelling Capability - Enumerated Scheduling

- Mechanism for parameterising processing based on vector or matrix dimensions
 - Not always possible - can depend on the processing being carried out
- Can eliminate dynamic scheduling overhead incurred by variable-sized processing
- When used in conjunction with families, can result in greatly simplified canvases
 - Improves maintainability, clarity of design and understandability
- Used extensively in CAPTOR as many modes support multiple data-set sizes



Improvements in Modelling Capability - Segmentation

- Runtime delimitation of subsets of data
- Can run specific processing at start or end of segment
- Scope extends only to next box (flowgraph or primitive)
- Segments processed in strict order
 - Provides flow control and current state of application
- Enables other features such as state and exclusion



```

Name: segementer
Type: static
Input: {
    stream int in;
    int Split;
}
Output: {
    segmented dynamic stream int out;
}
Apply: {
    int g;
    int count = 0;
    for (g = 0; g < granularity; g++) {
        out[g] = in[g];
        count++;
        if (out[g] == Split) {
            produce(out, count);
            segment(out, SEGMENT_END);
            segment(out, SEGMENT_BEGIN);
            count = 0;
        }
    }
    produce(out, count);
}
    
```

```

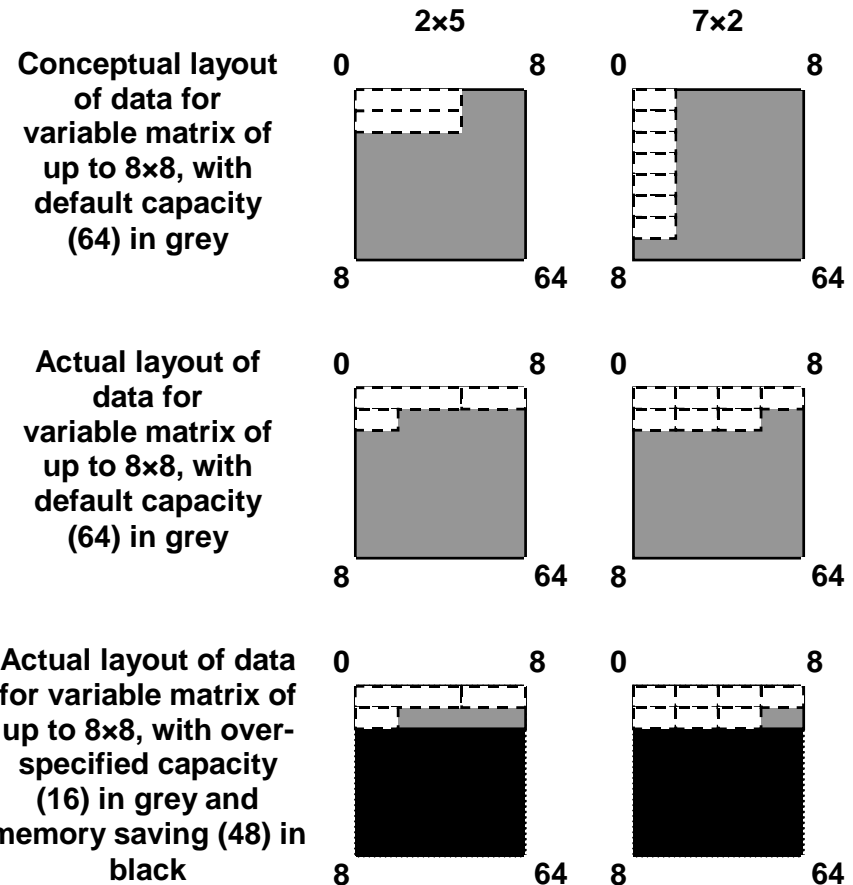
Name: seg_sum
Type: static
Input: {
    stream int in;
}
Local: {
    int sum;
}
Output: {
    dynamic int out;
}
Reset: {
    sum = 0;
}
Apply: {
    int g;
    for (g = 0; g < granularity; g++) {
        sum += in[g];
    }
}
EndOfSegment: {
    *out = sum;
    produce(out, 1);
}
    
```

Improvements in Modelling Capability - External State

- Many different ways of implementing state data
- Not all appropriate when this data must be accessed from distributed processors
 - Must ensure that correct data is available on each processor when required
 - Must enforce strict “order of access” control to ensure correct behaviour
- External state satisfies these requirements by keeping track of which segment is being processed on which processor, and moving the state data to the next processor on which it is required
 - Gedae automatically generates and manages all connections required to propagate and maintain state data
 - These implicit connections do not appear on the canvas, making it considerably less cluttered than an equivalent explicitly connected one
- Hierarchical position of the state box with regard to the segmenter determines its reset behaviour
 - Used in CAPTOR to control state reset across groups of related modes

Improvements in Memory Use - Over-Specified Variable Matrices

- Variable matrices allow dimensions of data to be specified at runtime, within compile-time defined limits
- To accommodate all data-set sizes, compile-time limits must be large in both dimensions, even when maximum volume of data is small
- Actual memory required to hold data is small, as data is stored packed
- Over-specification allows maximum capacity of variable matrix to be defined
 - Designer must ensure this capacity is never exceeded
 - Default capacity is height x width
- Considerable reductions achieved

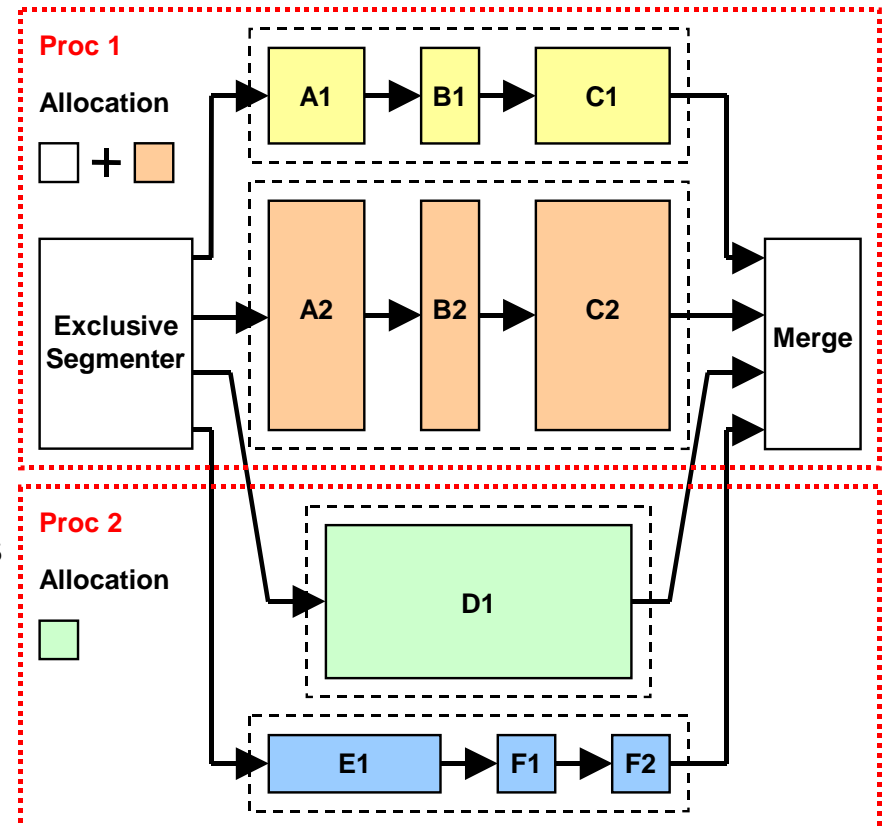


Improvements in Memory Use - Externally Allocated Schedule Memory

- Data I/O is usually processor-intensive
 - Particularly costly where very high data rates must be maintained
 - Significant time spent on I/O for some T2 modes due to data-set sizes
- Due to Gedae's pre-allocation of schedule memory and the nature of the low-level CAPTOR software facilities, two copy operations would have normally been required to make data available to the application
 - from the I/O buffer to the processor's dynamically-allocated heap
 - from there to the application's schedule memory
- Allowing externally allocated memory to be used as schedule memory eliminates the need for the latter copy operation
 - Saves both time and memory
 - Simply need to set stream pointer to address of externally allocated memory
 - Fast and simple to implement multiply-buffered I/O - reduces latency
 - Gedae automatically frees the external memory when no longer required

Improvements in Memory Use - Exclusion

- Families of segmented outputs may be flagged as exclusive
 - Only one of these family members (*i.e.* the entire flowgraph or primitive) at a time can process any part of a segment on any single processor
 - Memory need only be allocated for the largest of the family members on each processor
 - On a single processor, memory is thus traded off against latency
- Due to required behaviour of T2 SPS, all variants of all modes on a processor are exclusive
 - Saves approximately 85% of memory required, compared to non-exclusive implementation



Improvements in Application Executable Generation - Launch Package Components

- A Launch Package contains all of the code and data that a Gedae graph requires, without any dependency on the development GUI
- Many options available to configure how the various components are dealt with
 - Not all available to CAPTOR due to limited amount of non-volatile memory and rapid start-up time required
- Need to minimise size of overall launch package - start with individual components
 - Eliminate redundant and duplicate fields
 - Remove static zero-initialised data, change start-up sequence to allocate and initialise it instead
- Nature of T2 SPS (multiple instances of multiple modes, with multiple variants, connected into framework in a reasonably symmetrical fashion) → high commonality
 - Generate common libraries / components - smaller than the sum of their parts
 - Tailored when local copy unpacked to separate target processors
 - Results in approximately 30% reduction in size of launch package

Improvements in Application Executable Generation - Compression

- Two main targets for compression - individual components and final executable
 - Components re-arranged to make data more amenable to internal compression
 - Final executable then subjected to further external compression
- Flexible solution implemented, allowing users to specify own (external) compression algorithm
 - Uses gzip by default - good results
- Must trade off reduction in size against time taken to unpack compressed data
 - Time to compress unimportant, as this will be only at build time
 - Must include size of decompression executable in final launch package
- Final results not yet available
 - Initial compression evaluation showed 90% reduction in size, although less expected for final application
 - No timing analysis performed yet, but scope for trading off size versus speed

Conclusions

- Gedae's capabilities have greatly improved since its original selection for CAPTOR
- Many new features, only those of direct relevance to CAPTOR covered here
- These have been developed in a generically useful way, of benefit to both Gedae and CAPTOR
 - Gedae has become even more suitable for use on similarly large or complicated projects
 - Improved modelling capability
 - Better support for resource-limited designs
 - CAPTOR has taken full advantage of these advanced, esoteric features, to maximise the benefits of using Gedae
 - Clear, efficient design
 - On target to meet strict Eurofighter Typhoon radar requirements

- Any Questions?