

1 Register Construction

Parallel registers are characterized by four properties: number of readers, number of writers, number of register values, and the consistency guarantees that the register can give. All of these types of registers can be constructed from one another by specific register constructions. The previous lecture covered the construction of:

- MRSW from SRSW, for any combination of k-ary or binary, regular or safe
- MRSW, k-ary Safe from Binary Safe
- MRSW, k-ary Regular from Binary Regular
- MRSW, k-ary Atomic from Binary Atomic
- Binary Regular from Binary Safe

1.1 Resource metrics

Some of the constructions have required a large number of component registers to produce a register. Metrics to describe the use of component registers are therefore useful to gauge how efficiently an algorithm \(A\) that implements a register of type \(X\) using registers of type \(X\).

We define 3 measures for a register implementation algorithm \(A\):

- \(M_A\) the number of \(X\) registers required to compose one \(X\) register (memory)
- \(R_A\) the number of operations on \(X\) registers in one read operation of the \(X\) register
- \(W_A\) the number of operations on \(X\) registers in one write operation of the \(X\) register

We will usually consider the maximum value across any execution sequence when describing values for \(R_A\) and \(W_A\).

1.2 k-ary regular from Binary Regular

The algorithm for implementing a k-ary regular register from binary regular registers was introduced in the previous lecture. The implementation is:
INIT Let $x_v = 1$ and $\forall i \neq v : x_i = 0$

READ
1: for $i = 1$ to $k$ do
2: if $x_i = 1$ then
3: Return $i$
4: end if
5: end for

WRITE($v$)
1: Set $x_v = 1$
2: for $i = v - 1$ down to 0 do
3: Set $x_v = 0$
4: end for

Claim 1 A read will always return the value of the latest write that it reads from.

Proof: Let $W(v)$ be the latest write that a read $R$ reads from, and let $x_i$ be the last value written by $W$ that $R$ reads. If $x_i = 1$, then $i = v$ the read returns $i$. If $x_i = 0$, then $i < v$ (otherwise the writer would not have written to $x_i$) and the reader continues to read $x_{i+1}$. This register was also written to by $W$, so if the reader does not read the value written by $W(v)$, it must have read a value written by a later write $W'$. This contradicts the statement that $W$ was the latest write that $R$ reads from.

Claim 2 The above algorithm implements a regular register.

Proof: Let $W$ be the last write that completes before the start of a read $R$. By the algorithm, the write $W$ must have modified $x_0$, and the read $R$ must have read $x_0$ after $W$ wrote to it. By claim 1, this implies that the read cannot return the value of a write that preceded $W$. Because $W$ was the last preceding write, this means that the value returned by $R$ must be the value of that write, or an overlapping write; this is the definition of a regular register.

This algorithm takes $M = R = W = k$.

1.3 Lower Bound – Memory, Reads

There exist lower bounds on the values for $M, R, W$ in implementing a register of one type using registers of another. Consider algorithms that implement a $k$-ary safe register using binary safe registers.
Theorem 3 Any algorithm implementing a $k$-ary safe register using binary safe registers will have $M_A \geq \lceil \log_2 k \rceil$ and $R_A \geq \lceil \log_2 k \rceil$.

Proof: When the system is in a quiescent state, the reader must be able to distinguish between the $k$ states using the information it available in the register. The smallest number of bits that can distinguish between $k$ values is $\lceil \log_2 k \rceil$, so at least this many bits must be present to store the value in the register.

Suppose an algorithm $A$ exists that runs less than $\log_2 k$ operations for any read. Consider the execution of $READ(u)$ and $READ(v)$ with $u \neq v$. This execution consists of a number of lower-level read operations, and two high-level reads can return different values only if they read different values during the some low-level read operation. Let $i$ be the first read operation that the executions of $READ(u)$ and $READ(v)$ differ. By using the pigeonhole principle, $k$ values requires that $i \geq \lceil \log_2 k \rceil$ for some pair of values $u, v$.

1.4 Lower Bound – Writes

Unlike readers, which can only use the values of registers to get information from a system, writers can also use the identity of a register as an information channel. This allows the lower bound on $W_A$ to be much looser. The following algorithm has a $W_A = 2$ using $k$ registers $X_1, \ldots, X_k$

\begin{verbatim}
INIT(v)  X_v = 1 and \forall i \neq v : X_i = 0
WRITE(v)  1: Let $u$ be the current value (read from local processor storage)
           2: $X_u := 0$
           3: $X_v := 1$
READ
           1: for $i = 1$ to $k$ do
               2: if $X_i = 1$ then
                   3: Return $i$
               4: end if
           5: end for
           6: Return some arbitrary valid value $i$
\end{verbatim}

Lemma 4 After the end of any $WRITE(v)$, the registers are in the state described by $INIT(v)$.
Proof: As reads do not change the values of any register, they can be disregarded. Also, since the writes are done by a single processor, no two writes are overlapping. Let $W$ be the first write after which the claim does not hold. Before the start of the write, the system is in the state $INIT(u)$ with $u$ being the current value of the register. After step 2 completes, $X_i = 0$ for all $i \in [1,k]$. After step 3 complete, $\forall i \neq v : X_j = 0$ and $X_v = 1$, which is the description of $INIT(v)$. Therefore the claim holds across the write $W$ and so must hold across every write.

Claim 5 The register described above is a safe $k$-ary register with $(R,M,W) = (k,k,2)$

Proof: The bounds on the read, write, and memory use are clear from a simple inspection of the algorithm. Let $R$ be a read that does not overlap any write. By lemma 4, we know that the value of the register is $INIT(v)$ where $v$ is the value of the previous write. During the read, step 2 will test $X_v = 1$ and return $v$. No other value $v'$ can be returned at step 3 by the read because that would contradict the description of $INIT(v)$ which states that $X_{v'} = 0$.

Note that the register described above is not regular; it is possible for a reader to read all $X_i$ as zero if the writer decreases the value while the reader was reading.

There exists an algorithm called the “hypercube algorithm” with $W_A = 1$ and $R_A = M_A = k - 1$ for all $k = 2^l, l \in \mathbb{N}$.

1.5 Combined Lower Bound for $W$ and $M$

There is a trade-off between the number of writes and the number of memory cells. In particular, the first algorithm described and the hypercube algorithm illustrate this trade-off. This can be formalized by the theorem below. This theorem was not discussed in class but is the contribution of the scribe.

Theorem 6 Given values for $M_A$ and $W_A$, at most $k$ distinct values can be written to the resulting combined register where

$$k \leq \sum_{i=0}^{W_A} \binom{M_A}{i} = \sum_{i=0}^{W_A} \frac{M_A!}{i!(M_A - i)!}$$

Proof: Let $S$ be the state of the system before the write. If the register is able to take $k$ different values, then there must be $k$ possible destination states after the write completes. Let $W$ be a write to the register, and let $S'$ be the state after the write completes. Let $i$ be the number of bits that differ between states $S$ and $S'$. Because an operation in the write can change only one bit, $i \leq W_A$. For a given state $S$, there are exactly $\binom{M_A}{i}$ possible states $S'$ that differ from $S$ by $i$ bits. Therefore, $\sum_{i=0}^{W_A} \binom{M_A}{i}$ states can be reached from $S$, so the number of distinct values $k$ is at most this sum.

This bound is tight for both the $(W_A, M_A) = (1,k - 1)$ and $(W_A, M_A) = (\log_2 k, \log_2 k)$ algorithms discussed for implementing a $k$-ary safe register.